ΠΑΝΕΠΙΣΤΗΜΙΟ
ΘΕΣΣΑΛΙΑΣ

ΣΧΟΛΗ ΘΕΤΙΚΩΝ ΕΠΙΣΤΗΜΩΝ

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ

Ανάπτυξη Ηλεκτρονικού Παιχνιδιού με Ενσωματωμένο Σύστημα Επιλογής Κινήσεων

Μπαμπαράκος Νικόλαος

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

ΥΠΕΥΘΥΝΟΣ

ΚΟΛΟΜΒΑΤΣΟΣ ΚΩΝΣΤΑΝΤΙΝΟΣ
ΑΝΑΠΛΗΡΩΤΗΣ ΚΑΘΗΓΗΤΗΣ

Λαμία έτος 2025

ΠΑΝΕΠΙΣΤΗΜΙΟ ΘΕΣΣΑΛΙΑΣ

ΣΧΟΛΗ ΘΕΤΙΚΩΝ ΕΠΙΣΤΗΜΩΝ

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ

Ανάπτυξη Ηλεκτρονικού Παιχνιδιού με Ενσωματωμένο Σύστημα Επιλογής Κινήσεων

Μπαμπαράκος Νικόλαος

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

ΥΠΕΥΘΥΝΟΣ

ΚΟΛΟΜΒΑΤΣΟΣ ΚΩΝΣΤΑΝΤΙΝΟΣ
ΑΝΑΠΛΗΡΩΤΗΣ ΚΑΘΗΓΗΤΗΣ

Λαμία έτος 2025

UNIVERSITY OF
THESSALY

SCHOOL OF SCIENCE

DEPARTMENT OF COMPUTER SCIENCE & TELECOMMUNICATIONS

Development of a Video Game with an Integrated Move Selection System

Mpamparakos Nikolaos

FINAL THESIS

ADVISOR

KOLOMVATSOS KONSTANTINOS
ASSOCIATE PROFESSOR

Lamia year 2025

«Με ατομική μου ευθύνη και γνωρίζοντας τις κυρώσεις [(1)], που προβλέπονται από της διατάξεις της παρ. 6 του άρθρου 22 του Ν. 1599/1986, δηλώνω ότι:

1. Δεν παραθέτω κομμάτια βιβλίων ή άρθρων ή εργασιών άλλων αυτολεξεί ***χωρίς να τα περικλείω σε εισαγωγικά*** *και χωρίς να αναφέρω το συγγραφέα, τη χρονολογία, τη σελίδα. Η αυτολεξεί παράθεση χωρίς εισαγωγικά χωρίς αναφορά στην πηγή, είναι λογοκλοπή. Πέραν της αυτολεξεί παράθεσης, λογοκλοπή θεωρείται και η παράφραση εδαφίων από έργα άλλων, συμπεριλαμβανομένων και έργων συμφοιτητών μου, καθώς και η παράθεση στοιχείων που άλλοι συνέλεξαν ή επεξεργάσθηκαν, χωρίς αναφορά στην πηγή. Αναφέρω πάντοτε με πληρότητα την πηγή κάτω από τον πίνακα ή σχέδιο, όπως στα παραθέματα.*

2. Δέχομαι ότι η αυτολεξεί ***παράθεση χωρίς εισαγωγικά****, ακόμα κι αν συνοδεύεται από αναφορά στην πηγή σε κάποιο άλλο σημείο του κειμένου ή στο τέλος του, είναι αντιγραφή. Η αναφορά στην πηγή στο τέλος π.χ. μιας παραγράφου ή μιας σελίδας, δεν δικαιολογεί συρραφή εδαφίων έργου άλλου συγγραφέα, έστω και παραφρασμένων, και παρουσίασή τους ως δική μου εργασία.*

3. Δέχομαι ότι υπάρχει επίσης περιορισμός στο μέγεθος και στη συχνότητα των παραθεμάτων που μπορώ να εντάξω στην εργασία μου εντός εισαγωγικών. Κάθε μεγάλο παράθεμα (π.χ. σε πίνακα ή πλαίσιο, κλπ), προϋποθέτει ειδικές ρυθμίσεις, και όταν δημοσιεύεται προϋποθέτει την άδεια του συγγραφέα ή του εκδότη. Το ίδιο και οι πίνακες και τα σχέδια

4. Δέχομαι όλες τις συνέπειες σε περίπτωση λογοκλοπής ή αντιγραφής.

Ημερομηνία: ……/…../20……

Ο – Η Δηλ.

(1) «Όποιος εν γνώσει του δηλώνει ψευδή γεγονότα ή αρνείται ή αποκρύπτει τα αληθινά με έγγραφη υπεύθυνη δήλωση
του άρθρου 8 παρ. 4 Ν. 1599/1986 τιμωρείται με φυλάκιση τουλάχιστον τριών μηνών. Εάν ο υπαίτιος αυτών των πράξεων
σκόπευε να προσπορίσει στον εαυτόν του ή σε άλλον περιουσιακό όφελος βλάπτοντας τρίτον ή σκόπευε να βλάψει άλλον, τιμωρείται με κάθειρξη μέχρι 10 ετών.»

ΠΕΡΙΛΗΨΗ

Αντικείμενο της παρούσας πτυχιακής εργασίας είναι η μεθοδική ανάπτυξη ενός ηλεκτρονικού παιχνιδιού ρόλων. Η εργασία εξετάζει αρχικά τους περιορισμούς και τις διαδικασίες για τη λήψη θεμελιωδών σχεδιαστικών αποφάσεων. Στη συνέχεια, αναλύει εκτενώς τα συστατικά και τους μηχανισμούς που απαρτίζουν το παιχνίδι. Η διαδικασία αυτή ολοκληρώνεται με την υλοποίηση ενός πρωτοτύπου λογισμικού στη μηχανή παιχνιδιών Godot, με χρήση της γλώσσας προγραμματισμού C#.

Επιπλέον, σχεδιάζεται και υλοποιείται ένα σύστημα επιλογής κινήσεων, το οποίο βασίζεται στον αλγόριθμο Expectiminimax. Το σύστημα αυτό ενσωματώνεται στο πρωτότυπο, ελέγχοντας έναν χαρακτήρα-βοηθό του παίκτη και επιλέγοντας αυτόνομα τη βέλτιστή ενέργεια για λογαριασμό του.

ABSTRACT

The objective of this thesis is the methodical development of a Role-Playing Game (RPG). The work first examines the constraints and procedures for making fundamental design decisions. Subsequently, it provides an extensive analysis of the components and mechanics that constitute the game. This process concludes with the implementation of a software prototype in the Godot game engine, using the C# programming language.

Furthermore, a move selection system based on the Expectiminimax algorithm is designed and implemented. This system is integrated into the prototype, controlling a companion character for the player and autonomously selecting the optimal action on its behalf.

Table of Contents

ΚΕΦΑΛΑΙΟ 1- Εισαγωγή

1.1. Σκοπός και Δομή της Πτυχιακής

Ο σκοπός της παρούσας πτυχιακής εργασίας είναι ο σχεδιασμός και υλοποίηση ενός ηλεκτρονικού παιχνιδιού ρόλων, καθώς και η ενσωμάτωση ενός υπο-συστήματος τεχνητής νοημοσύνης, το οποίο είναι υπεύθυνο για τη λήψη αποφάσεων εντός του εικονικού κόσμου του παιχνιδιού.

Η εργασία δομείται σε πέντε κεφάλαια:

Κεφάλαιο 1- Εισαγωγή: Η εισαγωγή των βασικών εννοιών που θα απασχολήσουν τον αναγνώστη.

Κεφάλαιο 2- Θεωρητικό Πλαίσιο: Η εμβάθυνση των εννοιών που συστήθηκαν στην εισαγωγή, καθώς και πιο συγκεκριμένων στοιχείων που θα χρησιμοποιηθούν στα ακόλουθα κεφάλαια.

Κεφάλαιο 3- Σχεδιασμός και Ανάπτυξη του Ηλεκτρονικού Παιχνιδιού: Το κεφάλαιο αυτό αποτελείται από τρεις ενότητες, κάθε μια από αυτές αντικατοπτρίζει μια από τις φάσεις ανάπτυξης ενός ηλεκτρονικού παιχνιδιού. Η ενότητα δύο που αποτελεί την μεγαλύτερη σε εύρος ενότητα χωρίζεται σε δύο υπο-ενότητες, με την πρώτη να εστιάζει στον αλγοριθμικό σχεδιασμό του παιχνιδιού και την δεύτερη στην υλοποίηση του μέσω κώδικα.

Κεφάλαιο 4- Ενσωμάτωση της Τεχνητής Νοημοσύνης: Ο αλγοριθμικός σχεδιασμός του συστήματος που θα χρησιμοποιηθεί στο παιχνίδι για την λήψη αποφάσεων, καθώς και η υλοποίηση του σε κώδικα.

Συμπεράσματα: Ο γενικός απολογισμός της εργασίας, που θα επικεντρώνεται στο πόσο αποδοτικά βοηθάει το ενσωματωμένο σύστημα τεχνητής νοημοσύνης τον παίκτη στην επίτευξη της νίκης.

1.2. Ορισμοί: Παιχνίδια και Ηλεκτρονικά Παιχνίδια

Οι *Katie Salen* και *Eric Zimmerman* ορίζουν ένα παιχνίδι ως "ένα σύστημα στο οποίο οι παίκτες συμμετέχουν σε μια τεχνητή σύγκρουση, η οποία ορίζεται από κανόνες και οδηγεί σε ένα μετρήσιμο αποτέλεσμα"[1] . Αποτελώντας μια από τις κυριότερες μορφές ψυχαγωγίας, τα παιχνίδια χαρακτηρίζονται από αξιοσημείωτη πολυμορφία, καθώς η έννοια τους εκτείνεται από δραστηριότητες όπως το σκάκι, έως και αθλητικούς αγώνες.

Η μορφή παιχνιδιών που θα εξεταστεί στην παρούσα πτυχιακή εργασία είναι τα ηλεκτρονικά παιχνίδια. Αυτό που διακρίνει τα ηλεκτρονικά παιχνίδια από τις υπόλοιπες μορφές παιχνιδιών δεν είναι η απόρριψη των βασικών αρχών τους, αλλά το μέσο στο οποίο υλοποιούνται και αναπαράγονται. Σε ένα μη-ηλεκτρονικό παιχνίδι, η τεχνητή σύγκρουση μεταξύ των παικτών λαμβάνει μέρος εντός ενός υλικού πλαισίου, όπου οι κανόνες και η κατάσταση του παιχνιδιού ελέγχονται από τους ίδιους τους παίκτες ή ένα τρίτο πρόσωπο, συνήθως κάποιον διαιτητή. Αντιθέτως, στην περίπτωση των ηλεκτρονικών παιχνιδιών, το παιχνίδι λαμβάνει μέρος και ελέγχεται από ένα υπολογιστικό σύστημα.

1.3. Ιστορική Αναδρομή: Η Δημιουργία και η Εξάπλωση των Ηλεκτρονικών Παιχνιδιών

Η ραγδαία εξέλιξη της τεχνολογίας λογισμικού και υλικού σε συνδυασμό με τη τάση για την αξιοποίηση της πέρα από ερευνητικούς σκοπούς αποτέλεσε στην δημιουργία και εξάπλωση των πρώτων ηλεκτρονικών παιχνιδιών. Όπως υποστηρίζει ο *Tristan Donavan*

στο βιβλίο του "***Replay: The History of Games***" η διαδικασία αυτή δεν ήταν στιγμιαία, αλλά μια σταδιακή πορεία, η οποία χαρακτηρίστηκε από πολλούς πειραματισμούς, αποτυχίες και ευτυχείς συγκυρίες [2].

Η μορφή των πρώτων ηλεκτρονικών παιχνιδιών διέφερε ριζικά με αυτή των σημερινών, καθώς η ανάπτυξη τους γινόταν αποκλειστικά σε ακαδημαϊκά περιβάλλοντα και συνήθως ήταν απόπειρες για την μεταφορά υπαρχόντων παιχνιδιών σε ηλεκτρονική μορφή. Μια από αυτές τις πρώιμες απόπειρες έγινε από τον *Alan Turing* το 1947 , ο οποίος προσπάθησε να δημιουργήσει έναν αλγόριθμο που θα επέτρεπε σε έναν υπολογιστή να παίζει σκάκι εναντίον ανθρώπων. Τα υπολογιστικά συστήματα της εποχής όμως δεν ήταν αρκετά προηγμένα ώστε να μπορέσει να εφαρμοστεί [2].

Άλλη μια απόπειρα μεταφοράς ενός κλασικού παιχνιδιού σε υπολογιστή έγινε από τον *Arthur 'Art' Samuel*, που επιχείρησε να δημιουργήσει ένα πρόγραμμα υπολογιστή που να παίζει *Checkers*. Η πρώτη εκδοχή του προγράμματος ολοκληρώθηκε το 1952 σε έναν *IBM 701* και με συνεχείς βελτιώσεις μέχρι το 1961 ήταν ικανό να νικάει ανθρώπους [2].

Κομβικής σημασίας γεγονός για την ιστορία των ηλεκτρονικών παιχνιδιών είναι η δημιουργία του *Tennis for Two* το 1958 από τον *William Higinbotham* με την βοήθεια του μηχανικού *Robert Dvorak*. Ήταν ένα ηλεκτρονικό παιχνίδι που αναπαριστούσε έναν αγώνα τένις από μια πλάγια όψη του γηπέδου. Οι παίκτες χρησιμοποιούσαν χειριστήρια για να κινούν τις "ρακέτες" (λευκές γραμμές στην οθόνη) και πατώντας ένα κουμπί μπορούσαν να χτυπάνε την μπάλα. Θεωρείται ευρέως πως το *Tennis for Two* αποτελεί το πρώτο ηλεκτρονικό παιχνίδι με την σημερινή έννοια [2].

Το 1962, μια ομάδα φοιτητών του πανεπιστημίου *MIT*, αποτελούμενη από τους *Steve Russel*, *Wayne Witaenem* και *Martin Graetz*, αξιοποίησε την άφιξη ενός νέου υπολογιστή (*PDP-1*) που μόλις είχε εγκατασταθεί στο ίδρυμα για να δημιουργήσει το ηλεκτρονικό παιχνίδι *Spacewars!*. Στην τελική του μορφή, το παιχνίδι επέτρεπε σε δύο παίκτες να ελέγχουν αντίστοιχα διαστημόπλοια και να μονομαχούν σε ένα περιβάλλον που προσομοίωνε συνθήκες διαστήματος. Παρόλο που το *Spacewars!* δεν κυκλοφόρησε ποτέ σε εμπορική κλίμακα λόγω του υψηλού κόστους του απαιτούμενου εξοπλισμού, υπήρξε καθοριστική πηγή έμπνευσης για τα πρώτα ηλεκτρονικά παιχνίδια μαζικής παραγωγής, θέτοντας ουσιαστικά τις βάσεις για τη δημιουργία της αντίστοιχης βιομηχανίας [2].

Η δημιουργία του *Spacewars!* οδήγησε σε δύο ταυτόχρονες, αλλά ιδεολογικά διαφορετικές, προσπάθειες για την εμπορική του αξιοποίηση. Από τη μια πλευρά, οι φοιτητές του *Stanford*, *Bill Pitts* και *Hugh Tuck*, ανέπτυξαν μια σχεδόν πιστή μεταφορά του πρωτότυπου σε υπολογιστή *PDP-11*. Το αποτέλεσμα, το *Galaxy Game* (1971), κατέχει μεν τον τίτλο του πρώτου παιχνιδιού *arcade*, αλλά λόγω του ακριβού υλικού που απαιτούσε, δεν κατέστη εφικτό να παραχθεί μαζικά. Παράλληλα, οι *Nolan Bushnell* και *Ted Dabney* προσέγγισαν την ίδια ιδέα από μια πιο επιχειρηματική σκοπιά. Αποφεύγοντας την δαπανηρή διαδικασία της πιστής αναπαραγωγής του *Spacewars!*, οι δημιουργοί απλοποίησαν τους μηχανισμούς του και ανέπτυξαν οικονομικότερο, εξειδικευμένο υλικό για την υποστήριξη του. Το αποτέλεσμα, *Computer Space* (1971), αποτέλεσε το πρώτο εμπορικά διαθέσιμο ηλεκτρονικό παιχνίδι, επιβεβαιώνοντας έτσι τη δυνατότητα δημιουργίας ενός κερδοφόρου οικονομικού κλάδου [2].

Η υλοποίηση της ιδέας για την πρώτη οικιακή κονσόλα έλαβε εμπορική μορφή το 1972 με την κυκλοφορία του *Magnavox Odyssey*. Η συσκευή αποτελούσε την εμπορική εξέλιξη του πρωτοτύπου "*Brown Box*", το οποίο είχε αναπτύξει ο *Ralph Baer* μαζί με τους συνεργάτες του, *Bill Harrison* και *Bill Rusch*, στην εταιρεία *Sanders Associates*. Ο αντίκτυπος της κυκλοφορίας του *Odyssey* στην αγορά ήταν διπλός: όχι μόνο θεμελίωσε τον κλάδο των οικιακών ηλεκτρονικών παιχνιδιών, αλλά αποτέλεσε και άμεση πηγή έμπνευσης για την δημιουργία του *Pong* (1972), του παιχνιδιού που θα καθόριζε την επόμενη φάση της βιομηχανίας [2].

Μετά την ίδρυση της *Atari* το 1972 από τους *Nolan Bushnell* και *Ted Dabney*, το *Pong* αναδείχθηκε σε ένα από τα πρώτα εγχειρήματα που καθόρισαν την πορεία της εταιρείας.

Η έμπνευση για τη δημιουργία του παιχνιδιού πηγάζει απευθείας από το *Magnavox Odyssey*, καθώς ο *Bushnell*, εντυπωσιασμένος από το ενσωματωμένο παιχνίδι *Ping Pong* της κονσόλας, ανέθεσε στον νεαρό μηχανικό *Al Alcorn* τη δημιουργία μιας παρόμοιας έκδοσης ως εκπαιδευτική άσκηση. Ο *Alcorn*, ωστόσο, δεν περιορίστηκε στην απλή αναπαραγωγή, αλλά εισήγαγε βελτιώσεις στους μηχανισμούς που αναβάθμισαν σημαντικά την εμπειρία του παιχνιδιού. Η υψηλή ποιότητα του τελικού προϊόντος οδήγησε την *Atari* στην στρατηγική απόφαση να κυκλοφορήσει το παιχνίδι στα *arcades*. Η κίνηση αυτή αποτέλεσε την πρώτη μεγάλη εμπορική επιτυχία της εταιρείας, δίνοντας παράλληλα το έναυσμα για τον μετασχηματισμό της βιομηχανίας σε ένα κλάδο με αναγνωρισμένο πολιτιστικό και οικονομικό αντίκτυπο [2].

Η επιτυχία του *Pong* στα *arcades* προκάλεσε την άμεση εμφάνιση δεκάδων ανταγωνιστών που αντέγραφαν το μοντέλο της Atari. Η εμπορική αυτή πίεση, σε συνδυασμό με την αρχική αποτυχία των νέων τίτλων της, οδήγησε την εταιρεία σε σημαντικές οικονομικές δυσχέρειες. Ωστόσο, η μεγάλη εμπορική επιτυχία του παιχνιδιού *Tank* (1974) επέτρεψε στην *Atari* να επενδύσει στην ιδέα του μηχανικού *Harold Lee* για τη δημιουργία μιας οικιακής έκδοσης του *Pong*. Η υλοποίηση της κατέστη εφικτή χάρη σε ένα κρίσιμο τεχνολογικό επίτευγμα: την ενσωμάτωση ολόκληρου του παιχνιδιού σε ένα μόνο ολοκληρωμένο κύκλωμα, γεγονός που μείωσε δραστικά το κόστος παραγωγής. Η *Atari* εξασφάλισε μια αποκλειστική συμφωνία με την αλυσίδα *Sears* και τα Χριστούγεννα του 1975 κυκλοφόρησε την κονσόλα της, *Tele-Games Pong*, με τις πωλήσεις να ανέρχονται στις 150.000 μονάδες. Η κυκλοφορία της οικιακής έκδοσης του *Pong* δεν υπήρξε μόνο μια εμπορική επιτυχία, αλλά και ένα πολιτιστικό φαινόμενο, καθώς καθιέρωσε τα ηλεκτρονικά παιχνίδια ως μια δημοφιλή μορφή οικιακής ψυχαγωγίας και έθεσε τα θεμέλια για τη μετατροπή τους σε έναν βιομηχανικό κλάδο με ευρεία απήχηση και σημαντική οικονομική ισχύ [2].

1.4. Ηλεκτρονικά Παιχνίδια Ρόλων: Δημιουργία, Επιρροές και Βασικά Χαρακτηριστικά

Οι *Andrew Rollings* και *Ernest Adams* ορίζουν ένα παιχνίδι ρόλων ως "ένα παιχνίδι στο οποίο ο παίκτης αναλαμβάνει τον ρόλο ενός χαρακτήρα, και ως αυτός ο χαρακτήρας αυξάνει τη δύναμη και την ικανότητα του καθώς το παιχνίδι εξελίσσεται" [3]. Τα παιχνίδια ρόλων διακρίνονται κυρίως σε δύο βασικές κατηγορίες, την επιτραπέζια (*tabletop*) και την ηλεκτρονική. Αν και οι δύο αυτές κατηγορίες εμφανίστηκαν σχεδόν παράλληλα στην δεκαετία του 1970, τα ηλεκτρονικά παιχνίδια ρόλων κατέχουν τα δικά τους, διακριτά χαρακτηριστικά. Τα χαρακτηριστικά αυτά τα διαφοροποιούν όχι μόνο από τα επιτραπέζια παιχνίδια ρόλων αλλά και από τις υπόλοιπες κατηγορίες ηλεκτρονικών παιχνιδιών.

Ακολουθεί μια σύντομη διατύπωση των σημαντικότερων αυτών χαρακτηριστικών:

- Ο παίκτης δεν ελέγχει απρόσωπες ομάδες και στρατούς, αλλά έναν συγκεκριμένο χαρακτήρα ή μια μικρή ομάδα χαρακτήρων [4][5].
- Οι ικανότητες και οι αδυναμίες ενός χαρακτήρα αναπαρίστανται από ένα σύστημα ποσοτικών στατιστικών [4][5].
- Διαθέτουν ένα σύστημα προόδου, επιτρέποντας στα στατιστικά και τον εξοπλισμό του χαρακτήρα να βελτιώνεται διαρκώς [4][5].
- Η δράση τους πραγματοποιείται εντός ενός κόσμου που μπορεί να εξερευνηθεί [4][5].
- Περιέχουν αφήγηση που εξελίσσεται μέσω αποστολών (*quests*) και αλληλεπιδράσεων με μη-ελεγχόμενους χαρακτήρες *(Non-Playable Characters-NPCs)* [4][5].

- Το αποτέλεσμα των ενεργειών, και κυρίως της μάχης, καθορίζεται ως επί το πλείστων από τις στατιστικές τιμές του χαρακτήρα και όχι από τις φυσικές δεξιότητες του παίκτη [4][5].
- Η τήρηση των κανόνων και η αφήγηση αναλαμβάνονται πλήρως από τον υπολογιστή, γεγονός που καθιστά αδύνατο τον αυτοσχεδιασμό και την ερμηνεία απρόβλεπτων ενεργειών του παίκτη [4][5].

Στον πυρήνα τους, τα παιχνίδια ρόλων συνιστούν έναν συνδυασμό αφήγησης και ενός πολύπλοκου συστήματος κανόνων. Αυτή η διπλή φύση τους πηγάζει από μια πληθώρα επιρροών, προερχόμενων από διαφορετικά είδη ψυχαγωγίας. Οι σημαντικότερες από αυτές παρατίθενται παρακάτω [4][5].

Η πρώτη εμφάνιση πολύπλοκων συστημάτων κανόνων, παρόμοιων με εκείνα των παιχνιδιών ρόλων, ανιχνεύεται στα παιχνίδια πολέμου (*wargames*) που έκαναν την εμφάνιση τους στις αρχές του 19ου αιώνα. Το πρώτο παιχνίδι του είδους ήταν το *Kriegsspiel* (1812) του *Baron Von Reiswitz*, στο οποίο χρησιμοποιούνταν φιγούρες και ζάρια για την αναπαράσταση απρόσωπων μαχών μεταξύ στρατευμάτων. Τα παιχνίδια πολέμου συνέχισαν να εξελίσσονται, ενσωματώνοντας νέα χαρακτηριστικά όπως την εισαγωγή ενός διαιτητή. Ωστόσο, η κρίσιμη αλλαγή ήρθε στις αρχές της δεκαετίας του 1970, όταν παιχνίδια όπως το *Chainmail* (1970) του *Gary Gygax* εισήγαγαν κανόνες για συγκρούσεις μεταξύ ατομικών μονάδων. Αυτή η αλλαγή της εστίασης στις ατομικές μονάδες αποτέλεσε την κύρια έμπνευση για τα πρώτα συστήματα παιχνιδιών ρόλων [4][5].

Ενώ τα παιχνίδια πολέμου συνέβαλλαν σε σημαντικό βαθμό στη διαμόρφωση των συστημάτων κανόνων, η θεματική έμπνευση των παιχνιδιών ρόλων πηγάζει από το λογοτεχνικό ρεύμα της υψηλής φαντασίας (*High Fantasy*), και ειδικότερα από τα έργα του *J.R.R. Tolkien*. Η ραγδαία αύξηση της δημοτικότητας του "*Άρχοντα των Δαχτυλιδιών*" (1954) κατά τη δεκαετία του 1960 διαμόρφωσε τα βασικά αφηγηματικά πρότυπα του είδους. Συγκεκριμένα, καθιέρωσε τις συμβάσεις των αρχετυπικών φυλών, όπως τα ξωτικά, οι νάνοι και τα ορκ, καθώς και τη δομή της επικής αποστολής μιας ομάδας ηρώων ενάντια σε έναν παντοδύναμο εχθρό [4][5].

Το 1974 εκδόθηκε το *Dungeons & Dragons* από τους *Gary Gygax* και *Dave Arneson*, σηματοδοτώντας την εμφάνιση του πρώτου εμπορικά διαθέσιμου παιχνιδιού ρόλων. Το σύστημα κανόνων του βασίστηκε στους μηχανισμούς του *Chainmail*, ενώ η αφηγηματική του θεματολογία αντλήθηκε από τους κόσμους φαντασίας που θεμελίωσε ο *Tolkien*. Η κυριότερη καινοτομία του, ωστόσο, εντοπίζεται στον τρόπο δημιουργίας και εξέλιξης των χαρακτήρων. Αντίστρεψε την καθιερωμένη λογική των παιχνιδιών πολέμου, όπου η κλάση προκαθόριζε τις ικανότητες των μονάδων, επιτρέποντας στον παίκτη να ορίζει τα βασικά χαρακτηριστικά του ήρωα του και στη συνέχεια να επιλέγει την κατάλληλη κλάση. Ταυτόχρονα, το παιχνίδι καθιέρωσε τον ρόλο του *Dungeon Master* ως αφηγητή και διαιτητή, μια εξέλιξη του αντίστοιχου ρόλου στα παιχνίδια πολέμου. Μέσω αυτών των καινοτομιών, εδραιώθηκε το πρότυπο μιας ομάδας ηρώων με μοναδικά στατιστικά που αναλαμβάνει αποστολές, μια δομή που μεταφέρθηκε αυτούσια στα ηλεκτρονικά παιχνίδια ρόλων, με τον υπολογιστή να αναλαμβάνει τον ρόλο του *Dungeon Master* [4][5].

Η ταυτόχρονη κυκλοφορία του *Dungeons & Dragons* και η εμφάνιση προηγμένων υπολογιστικών συστημάτων σε πανεπιστήμια, όπως το PLATO IV (1972), αποτέλεσε το έναυσμα για τους φοιτητές της εποχής να πειραματιστούν με τη δημιουργία των πρώτων ηλεκτρονικών παιχνιδιών ρόλων. Ο ακριβής προσδιορισμός του πρώτου ηλεκτρονικού παιχνιδιού ρόλων είναι δύσκολος, εξαιτίας της πρακτικής των διαχειριστών των πανεπιστημιακών συστημάτων να διαγράφουν τα μη εξουσιοδοτημένα αρχεία. Ωστόσο, το παλαιότερο σωζόμενο και πλήρως λειτουργικό παιχνίδι του είδους είναι το *pedit5* (1975), με επίσημο τίτλο *The Dungeon*. Το παιχνίδι αυτό ενσωμάτωνε τα βασικά χαρακτηριστικά ενός ηλεκτρονικού παιχνιδιού ρόλων, όπως τη δημιουργία χαρακτήρων με στατιστικά, τις μάχες με τέρατα, τη συλλογή θησαυρών και ένα σύστημα προόδου βασισμένο σε πόντους εμπειρίας [4][5].

Έργα όπως το "*The Dungeon*" και οι άμεσοι διάδοχοι του απέδειξαν την επιτυχή μεταφορά του μοντέλου του *Dungeons & Dragons* σε ένα ψηφιακό μέσο, θεμελιώνοντας ένα είδος το οποίο, χάρη στο διαρκές ενδιαφέρον του κοινού, συνεχίζει να εξελίσσεται μέχρι και σήμερα [4][5].

1.5. Τα Στάδια Σχεδιασμού και Υλοποίησης ενός Ηλεκτρονικού Παιχνιδιού

Η ανάλυση σχετικά με την υλοποίηση και τον σχεδιασμό του ηλεκτρονικού παιχνιδιού θα βασιστεί στις αρχές και την ορολογία του συγγράμματος "***Fundamentals of Game Design***" του *Ernest Adams* [6].

Η επιλογή της συγκεκριμένης πηγής έγινε με βάση την ακαδημαϊκή της αναγνώρισης. Το βιβλίο αποτελεί ένα ευρέως αποδεκτό εγχειρίδιο στο τομέα του σχεδιασμού παιχνιδιών (*game design*), γεγονός που τεκμηριώνεται από τη χρήση του ως βιβλιογραφία σε προγράμματα πανεπιστημιακών μαθημάτων [7][8], καθώς και από τον σημαντικό αριθμό ακαδημαϊκών αναφορών που έχει λάβει στην πλατφόρμα *Google Scholar*.

Η δομή του τρίτου κεφαλαίου, στο οποίο θα αναπτυχθεί το ηλεκτρονικό παιχνίδι παραλληλίζει τα τρία στάδια της διαδικασίας σχεδίασης που ορίζει ο *Adams*:

Το στάδιο της σύλληψης της ιδέας (*concept stage*): Η αρχή της διαδικασίας. Ο σχεδιαστής καλείται να πάρει τις θεμελιώδεις αποφάσεις για το έργο του, όπως την κεντρική ιδέα, το κοινό στο οποίο απευθύνεται και τη γενικότερη κλίμακα του παιχνιδιού. Αυτές οι αποφάσεις παραμένουν σταθερές σε όλη τη διάρκεια της ανάπτυξης του παιχνιδιού [6].

Το στάδιο της επεξεργασίας (*elaboration stage*): Το στάδιο στο οποίο γίνεται εμβάθυνση των αρχικών αποφάσεων. Ο σχεδιαστής ορίζει τους κανόνες, τον πρωταγωνιστή και το βασικό *gameplay* του παιχνιδιού. Στην συνέχεια, βασιζόμενος στα παραπάνω δημιουργεί ένα σύνολο οντοτήτων για όλα τα στοιχεία του παιχνιδιού που απαιτούν επεξεργασία, καθώς και μηχανισμούς που ρυθμίζουν τις μεταξύ τους αλληλεπιδράσεις. Στο τέλος, οι μηχανισμοί αυτοί μεταφράζονται σε κώδικα και η συνολική διαδικασία αξιολογείται και επαναλαμβάνεται μέχρι να επιτευχθεί ένα επιθυμητό αποτέλεσμα. Σημαντική πρακτική σε αυτό το στάδιο είναι η δημιουργία πρωτοτύπων, τα οποία είναι μη-πλήρεις εκδοχές του παιχνιδιού και στοχεύουν στην αξιολόγηση συγκεκριμένων στοιχείων του ή της γενικής εμπειρίας πριν την επένδυση πόρων για ανάπτυξη [6].

Το στάδιο της βελτιστοποίησης (*tuning stage*): Αφού ολοκληρωθεί το στάδιο της επεξεργασίας, ο σχεδιαστής παύει να προσθέτει νέα στοιχεία. Επικεντρώνεται πλέον στην τελειοποίηση και την εξισορρόπηση των υπάρχοντών. Σκοπός είναι να γίνουν οι απαραίτητες προσαρμογές ώστε το παιχνίδι να προσφέρει μια εμπειρία που να είναι ταυτόχρονα δίκαιη και συναρπαστική για τον παίκτη [6].

Τέλος στο κεφάλαιο 4 θα γίνει μια ανάλυση στον σχεδιασμό και την υλοποίηση ενός συστήματος πρόβλεψης βέλτιστων κινήσεων το οποίο θα αξιοποιείται από ένα χαρακτήρα του παιχνιδιού.

Κεφάλαιο 2- Θεωρητικό Πλαίσιο

Το παρόν κεφάλαιο αποτελεί ένα θεωρητικό πλαίσιο για όλες τις βασικές και εξειδικευμένες έννοιες που πρόκειται να αναλυθούν στα ακόλουθα κεφάλαια. Το βασικό εγχείρημα του συνιστά την προετοιμασία του αναγνώστη για την πλήρη κατανόηση των περιεχομένων που παρουσιάζονται, διασφαλίζοντας ότι δεν θα απαιτηθεί επεξήγηση των ίδιων εννοιών στα επόμενα κεφάλαια.

2.1. Τα Στάδια Ανάπτυξης ενός Ηλεκτρονικού Παιχνιδιού

Σύμφωνα με τον *Adams*, η διαδικασία σχεδιασμού ενός ηλεκτρονικού παιχνιδιού είναι μια επαναληπτική διαδικασία που αποτελείται από τρία στάδια.

Αρχικά, το **στάδιο της ιδέας (*concept stage*)**, στο οποίο λαμβάνονται οι βασικές δημιουργικές αποφάσεις. Αυτό το στάδιο αποτελείται από τέσσερις βασικές δραστηριότητες:

- **Σύλληψη της ιδέας:** Είναι το αρχικό βήμα της διαδικασίας. Ο σχεδιαστής επιλέγει την κεντρική ιδέα του παιχνιδιού, η οποία καθορίζει τον τρόπο που σκοπεύει να ψυχαγωγήσει τον παίκτη. Βασικό κριτήριο αυτής της επιλογής αποτελεί το είδος στο οποίο θέλει να κατατάξει το παιχνίδι ο σχεδιαστής [6].
- **Ορισμός του κοινού:** Αφού καθοριστεί ο τύπος της εμπειρίας που πρόκειται να προσφέρει ο σχεδιαστής, πρέπει να προσδιοριστεί και το κοινό στο οποίο απευθύνεται. Η επιλογή του κοινού επηρεάζει τις μετέπειτα αποφάσεις που γίνονται κατά το στάδιο της επεξεργασίας, καθώς όλες οι αξιολογήσεις γίνονται με βάση αυτό το θεωρητικό κοινό. Σε ένα εμπορικό πλαίσιο, η επιλογή αυτή συνήθως προηγείται της ιδέας, διότι σε αυτή την περίπτωση το κοινό λειτουργεί ως "αγορά-στόχος" και η επιλογή του σχεδιαστή στοχεύει στην μεγιστοποίηση της εμπορικής επιτυχίας του παιχνιδιού [6].
- **Καθορισμός του ρόλου του παίκτη:** Σε ένα **αναπαραστατικό (*representational*) παιχνίδι**, η εμπειρία στηρίζεται στην πράξη της "προσποίησης" με τον παίκτη να ψυχαγωγείται από το παιχνίδι μεταξύ άλλων επειδή συνειδητά πιστεύει στον κόσμο και στις καταστάσεις που του παρουσιάζονται. Αντιθέτως, σε ένα **αφηρημένο (*abstract*) παιχνίδι**, ο ρόλος του είναι πιο περιορισμένος. Ένας σχεδιαστής είναι απαραίτητο να ορίσει με σαφήνεια τον ρόλο που έχει ο παίκτης στον φανταστικό κόσμο του παιχνιδιού ώστε να ενισχύσει τον βαθμό εμβύθισης του σε αυτόν [6].
- "**Εκπλήρωση του ονείρου":** Αυτή η μεταφορική ορολογία χρησιμοποιείται από τον *Adams* για να περιγράψει την υποχρέωση του σχεδιαστή να ανταποκριθεί στις προσδοκίες του παίκτη. Οι κόσμοι στους οποίους λαμβάνουν μέρος κυρίως τα αναπαραστατικά παιχνίδια είναι γνώριμοι στον παίκτη, με αποτέλεσμα να έχει κάποιες προσδοκίες σχετικά με τις προκλήσεις που αναμένει να αντιμετωπίσει και τις ενέργειες που θα μπορεί να κάνει. Ο σχεδιαστής καλείται να προσδιορίσει ποιες είναι αυτές οι προσδοκίες και να σχεδιάσει το παιχνίδι με στόχο να τις ικανοποιήσει [6].

Στην συνέχεια, ακολουθεί το **στάδιο της επεξεργασίας (elaboration stage)**, όπου ο σχεδιασμός μεταβαίνει από το θεωρητικό στο πρακτικό, μετατρέποντας τις αφηρημένες έννοιες σε συγκεκριμένες σχεδιαστικές αποφάσεις και εφαρμόσιμα στοιχεία. Αντίστοιχα με το προηγούμενο, και το παρόν στάδιο απαρτίζεται από μια σειρά δραστηριοτήτων:

- **Ορισμός του κύριου τρόπου παιχνιδιού:** Σε ένα παιχνίδι ενδέχεται να συνυπάρχουν πολλοί διαφορετικοί **τρόποι παιχνιδιού (*gameplay*)** ωστόσο, στις περισσότερες περιπτώσεις ένας ορίζεται ως ο κύριος. Για παράδειγμα, σε ένα παιχνίδι αγώνων αμαξιών, η οδήγηση αποτελεί τον κύριο τρόπο παιχνιδιού ενώ η τροποποίηση του αμαξιού σε ένα συνεργείο αποτελεί έναν δευτερεύον. Σε αυτή τη φάση του σχεδιασμού δεν είναι απαραίτητη η ρύθμιση όλων των λεπτομερειών, αλλά προτεραιότητα δίνεται στην ανάπτυξη των βασικών συστατικών που συνθέτουν τον κύριο τρόπο παιχνιδιού όπως η οπτική προοπτική του παίκτη (player perspective), το μοντέλο αλληλεπίδρασης (interaction model) του με τον κόσμο, οι προκλήσεις που καλείται να αντιμετωπίσει και οι ενέργειες που μπορεί να εκτελέσει για την αντιμετώπιση τους [6].
- **Σχεδιασμός του Πρωταγωνιστή:** Ο πρωταγωνιστής αποτελεί τον χαρακτήρα με τον οποίο ο παίκτης θα αλληλοεπιδράσει τον περισσότερο χρόνο κατά τη διάρκεια

του παιχνιδιού. Κρίνεται απαραίτητος ο σαφής καθορισμός των θεμελιωδών του χαρακτηριστικών, ειδικότερα σε παιχνίδια προοπτικής τρίτου προσώπου, καθώς οι εκφράσεις προσώπου, η γλώσσα του σώματος και ο τρόπος ομιλίας είναι διαρκώς εμφανή στον παίκτη [6].

- **Ορισμός του κόσμου του παιχνιδιού:** Με την ολοκλήρωση του σχεδιασμού του πρωταγωνιστή ακολουθεί ο καθορισμός του κόσμου στον οποίο εντάσσεται. Ο κόσμος αυτός απαρτίζεται από πολλές διαστάσεις, όπως η φυσική, η περιβαλλοντολογική, η συναισθηματική και η ηθική, οι οποίες πρέπει να καθοριστούν με σαφήνεια [6].
- **Σχεδιασμός των βασικών μηχανισμών:** Σύμφωνα με τον *Adams*, οι βασικοί μηχανισμοί είναι "τα δεδομένα και οι αλγόριθμοι που ορίζουν με ακρίβεια τους κεντρικούς κανόνες και τις εσωτερικές διαδικασίες του παιχνιδιού". Σε πρακτικό επίπεδο, αποτελούν το συμβολικό και μαθηματικό μοντέλο το οποίο μετατρέπει τους γενικούς κανόνες σε συγκεκριμένες, υλοποιήσιμες εντολές. Τα συστατικά των βασικών μηχανισμών αναλύονται εκτενώς σε επόμενη ενότητα (βλ. ενότητα 2.3)[6].
- **Δημιουργία επιπλέον τρόπων παιχνιδιού:** Σε πολλές περιπτώσεις κατόπιν του καθορισμού της κεντρικής ιδέας, προκύπτει η ανάγκη του σχεδιασμού επιπρόσθετων τρόπων παιχνιδιού για να συμπληρώνουν το κύριο. Σε αυτή τη φάση είναι δυνατός ο σχεδιασμός της προοπτικής, του μοντέλου αλληλεπίδρασης και της λειτουργίας τους. Η προσθήκη αυτών των επιπλέον τρόπων παιχνιδιών δεν πρέπει να είναι αυθαίρετη, αλλά να συμβαίνει μόνο αν συμβάλλουν στον εμπλουτισμό της συνολικής εμπειρίας [6].
- **Σχεδιασμός Επιπέδων:** Ο σχεδιασμός των επιπέδων είναι η διαδικασία δόμησης της εμπειρίας του παίκτη, αξιοποιώντας τα θεμελιώδη στοιχεία του παιχνιδιού, όπως οι χαρακτήρες, οι προκλήσεις, οι μηχανισμοί και η ιστορία. Η διαδικασία αυτή μπορεί να ξεκινήσει σε πρώιμο στάδιο της ανάπτυξης, ακόμη και πριν την τελική οριστικοποίηση όλων των στοιχείων. Προαπαιτείτε, ωστόσο, η διαθεσιμότητα ενός βασικού συνόλου από αυτά, ώστε να είναι εφικτή η έναρξη της [6].
- **Γράψιμο της ιστορίας:** Η ενσωμάτωση αφήγησης στο παιχνίδι αποτελεί έναν τρόπο ενίσχυσης του ενδιαφέροντος του παίκτη και παρέχει κίνητρο για περαιτέρω ενασχόληση. Η ιστορία είναι απαραίτητη σε παιχνίδια μεγαλύτερης κλίμακας ενώ σε μικρότερα παιχνίδια η παρουσία της είναι συχνά δευτερεύουσας σημασίας [6].
- **Ανάπτυξη, Αξιολόγηση και Επανάληψη**: Ο *Adams* αναφέρεται στον σχεδιαστή *Mark Cerny*, ο οποίος υποστηρίζει ότι τα ηλεκτρονικά παιχνίδια πρέπει να περνούν από το στάδιο της πρωτοτυποποίησης (βλ. ενότητα 2.2) πριν την τελική τους παραγωγή. Κατά συνέπεια, το στάδιο της ανάπτυξης δεν είναι γραμμικό, αλλά ένα επαναληπτικός κύκλος που περιλαμβάνει τη δημιουργία πρωτοτύπων, την αξιολόγηση τους από ομάδες δοκιμαστών και την επακόλουθη βελτίωση τους. Ο κύκλος αυτός συνεχίζεται μέχρι να επιτευχθεί το επιθυμητό αποτέλεσμα και οριστικοποιηθεί η τελική μορφή του παιχνιδιού [6].

Τέλος, το **στάδιο της ρύθμισης (tuning stage)**, το οποίο λαμβάνει χώρα κατά το τέλος του σταδίου της επεξεργασίας. Σε αυτό το σημείο, η προσθήκη νέου περιεχομένου έχει ολοκληρωθεί και ο κύριος στόχος είναι η λεπτομερής ρύθμιση των επιπέδων και των μηχανισμών, με απώτερο σκοπό τη διασφάλιση μιας ισορροπημένης και συνεκτικής εμπειρίας για τον παίκτη.

2.2. Έννοιες Πρακτικού Χαρακτήρα

Στην προηγούμενη ενότητα αναλύθηκαν οι διαδικασίες σχεδιασμού που έχουν κυρίως δημιουργικό χαρακτήρα. Η παρούσα ενότητα, αντίθετα, εστιάζει σε δύο έννοιες πρακτικής φύσης, οι οποίες συμβάλλουν στην ομαλή διεξαγωγή του έργου.

2.2.1. Εμβέλεια (Scope)

Το **εύρος (*scope*)** ορίζεται ως το συνολικό μέγεθος και η πολυπλοκότητα ενός έργου, περιλαμβάνοντας όλα τα χαρακτηριστικά, τα συστήματα και το περιεχόμενο του. Η εκτίμηση του εύρους αποτελεί μια κομβική διεργασία που πρέπει να πραγματοποιείται στα πρώιμα στάδια του σχεδιασμού, καθώς οριοθετείτε από συγκεκριμένους περιορισμούς, όπως το χρονοδιάγραμμα, τον προϋπολογισμό, την υπάρχουσα τεχνολογία και τις δυνατότητες της ομάδας ανάπτυξης. Ο σχεδιαστής οφείλει να βρει μια ισορροπία μεταξύ των δημιουργικών φιλοδοξιών του και των περιορισμών που του επιβάλλονται, αλλιώς υπάρχει μεγάλος κίνδυνός η διαδικασία να οδηγηθεί στο φαινόμενο του "***feature creep***". Κατά το φαινόμενο αυτό, η ανεξέλεγκτη προσθήκη νέων στοιχείων αυξάνει το κόστος και τον χρόνο ανάπτυξης, απειλώντας έτσι την επιτυχή ολοκλήρωση του έργου [9].

2.2. Πρωτοτυποποίηση (Prototyping)

Ένα **πρωτότυπο (*prototype*)** είναι μια απλοποιημένη, αλλά αξιολογήσιμη μορφή του παιχνιδιού. Ο σκοπός του είναι η εξέταση χαρακτηριστικών του παιχνιδιού πριν την δαπάνη πόρων για την πλήρη υλοποίηση τους, καθώς και η αξιολόγηση της μέχρι στιγμής εμπειρίας του παιχνιδιού από χρήστες-δοκιμαστές (*player-testers*). Στο σύγγραμμα τους, οι *Joris Dormans* και *Ernest Adams* προτείνουν την ακόλουθη διάκριση των πρωτοτύπων σε τρεις κατηγορίες:

- **Πρωτότυπα Λογισμικού (*Software Prototypes*):** Είναι μη ολοκληρωμένες, ψηφιακές εκδοχές του παιχνιδιού. Ως λογισμικό, προσφέρουν υψηλή πιστότητα (*high-fidelity*) στην αναπαράσταση της τελικής εμπειρίας και καθιστούν τη μετάβαση στο τελικό προϊόν ομαλότερη [10].
- **Χάρτινα Πρωτότυπα (*Paper Prototypes*):** Αποτελούν μη ηλεκτρονικές μορφές του παιχνιδιού, όπως επιτραπέζια και παιχνίδια με κάρτες. Τα χάρτινα πρωτότυπα είναι μια γρήγορη και οικονομική μέθοδος για να εξεταστούν οι κανόνες και οι μηχανισμοί του παιχνιδιού [10].
- **Φυσικά Πρωτότυπα (*Physical Prototypes*)**: Συνιστούν ένα μέσο για την εξέταση των φυσικών κινήσεων και αλληλεπιδράσεων που πρόκειται να εφαρμοστούν στο παιχνίδι. Συχνά αποτελούν προεργασία για διαδικασίες όπως το *motion capture* [10].

2.3. Δομή και Λειτουργία των Βασικών Μηχανισμών

Ο *Ernest Adams* ορίζει τους **βασικούς μηχανισμούς (*core mechanics*)** ενός παιχνιδιού ως "τα δεδομένα και τους αλγορίθμους που καθορίζουν με ακρίβεια τους κεντρικούς κανόνες και τις εσωτερικές λειτουργίες του παιχνιδιού". Στα πρώιμα στάδια της ανάπτυξης ενός ηλεκτρονικού παιχνιδιού, ο σχεδιαστής έχει μια αφηρημένη ιδέα σχετικά με τους κανόνες του έργου του. Ο σχεδιασμός μηχανισμών στοχεύει στην μετατροπή αυτών των γενικών κανόνων σε συγκεκριμένους και ολοκληρωμένους μηχανισμούς [6].

Για τον ακριβή προσδιορισμό των βασικών μηχανισμών, ο σχεδιαστής καλείται να προσδιορίσει ένα σύνολο διακριτών στοιχείων που ορίζουν την λειτουργία του παιχνιδιού του: πόρους, οντότητες, χαρακτηριστικά και μηχανισμούς [6]. Η παρούσα ενότητα εμβαθύνει σε αυτές τις έννοιες, με στόχο την περαιτέρω αξιοποίηση τους στα επόμενα κεφάλαια.

2.3.1. Τα Δομικά Στοιχεία των Μηχανισμών

Όπως επισημάνθηκε, οι μηχανισμοί αποτελούνται από αλγόριθμους και δεδομένα. Στην παρούσα υπο-ενότητα, θα προσδιοριστούν τα δομικά στοιχεία που έχουν τον ρόλο των δεδομένων ενός μηχανισμού [6].

Αρχικά, οι **πόροι (*resources*)** ορίζονται ως τα αντικείμενα τα οποία το παιχνίδι μπορεί να δημιουργεί, καταναλώνει, ανταλλάσσει και να καταστρέφει. Ο όρος "πόρος" δεν αντιπροσωπεύει τη μεμονωμένη μονάδα ενός αντικειμένου, αλλά τη συνολική, αριθμητική του ποσότητα. Αξίζει να σημειωθεί ότι οι πόροι μπορούν να έχουν και μη υλική υπόσταση, αντιπροσωπεύοντας αφηρημένες έννοιες όπως τη φήμη ενός χαρακτήρα [6].

Από την άλλη, οι **οντότητες (*entities*)** αναπαριστούν μια περίπτωση (*instance*) ενός πόρου ή μια κατάσταση ενός στοιχείου του κόσμου του παιχνιδιού. Ως οντότητες μπορούν να προσδιοριστούν ένας χαρακτήρας, ένα κτήριο ή η κατάσταση ενός φαναριού σε μια δεδομένη χρονική στιγμή. Οι οντότητες κατηγοριοποιούνται με βάση τον τύπο τιμής που λαμβάνουν

- Μια **αριθμητική οντότητα (*numeric entity*)** λαμβάνει τιμές που είναι μετρήσιμες και μπορούν να υποστούν μαθηματική επεξεργασία [6].
- Αντίθετα, μια **συμβολική οντότητα (*symbolic entity*)** λαμβάνει τιμές που προσδιορίζουν τη διακριτή της κατάσταση στο παιχνίδι. Αυτές οι τιμές δεν μπορούν να υποστούν μαθηματική επεξεργασία. Υπο-κατηγορία των συμβολικών οντοτήτων αποτελούν οι **σημαίες (*flags*)** με διαχωριστικό χαρακτηριστικό ότι έχουν μόνο δύο καταστάσεις.

Επιπλέον, οι οντότητες διακρίνονται περαιτέρω σε απλές και σύνθετες, με βάση το πλήθος των τιμών που απαιτούνται για τον προσδιορισμό τους:

- Μια **απλή οντότητα (*simple entity*)** προσδιορίζεται μόνο από μια τιμή. Αυτή η τιμή μπορεί να είναι είτε αριθμητική είτε συμβολική [6].
- Μια **σύνθετη οντότητα (*compound entity*)** περιγράφεται από ένα σύνολο άλλων οντοτήτων. Αυτές οι οντότητες ονομάζονται **χαρακτηριστικά (*attributes*)** της σύνθετης οντότητας, οι οποίες με τη σειρά τους μπορούν να είναι επίσης σύνθετες [6].

Για λόγους συνέπειας και καλύτερης αναγνωσιμότητας στην ανάλυση του πρακτικού σχεδιασμού των μηχανισμών και στην υλοποίηση τους, υιοθετείται μια συγκεκριμένη σύμβαση ονοματοδοσίας. Όλες οι οντότητες, οι πόροι και τα χαρακτηριστικά θα αναφέρονται με τα αγγλικά τους ονόματα (π.χ. *Player*, *HP*). Επιπλέον, η αναφορά σε ένα συγκεκριμένο χαρακτηριστικό μιας οντότητας θα ακολουθεί τη δομή "*Όνομα_Οντότητας.Όνομα_Χαρακτηριστικού*" (π.χ. *Player.HP*). Τέλος, όταν η περιγραφή ενός μηχανισμού αναφέρεται σε ένα χαρακτηριστικό που αφορά ταυτόχρονα πολλές οντότητες ίδιου τύπου (όπως ο επιτιθέμενος και ο αμυνόμενος σε μια μάχη), θα χρησιμοποιείται η δομή "*Χαρακτηριστικό_Ομάδα_Οντοτήτων*" (π.χ. *ATK_επιτιθέμενου*).

2.3.2. Ο Τρόπος Εκτέλεσης των Μηχανισμών και η Εσωτερική Οικονομία

Οι μηχανισμοί του παιχνιδιού προσδιορίζουν τις σχέσεις και τις αλληλεπιδράσεις των προαναφερθεισών δομικών στοιχείων. Αυτές οι σχέσεις μπορεί να είναι αριθμητικές, όπως μια εξίσωση που υπολογίζει έναν αριθμό ζημιάς ή συμβολικές, όπως ένας κανόνας που ορίζει τον τρόπος με τον οποίο η κατάσταση "παγωμένος" επηρεάζει μια κίνηση. Η λειτουργία των μηχανισμών εκφράζεται μέσω **γεγονότων (*events*)** και **διαδικασιών (*processes*)**, τα οποία ενεργοποιούνται από συγκεκριμένες **συνθήκες (*conditions*)**:

- **Γεγονότα (*Events*):** Αποτελούν αλλαγές που συμβαίνουν στο κόσμο του παιχνιδιού όταν ενεργοποιηθεί μια συνθήκη και δεν επαναλαμβάνονται μέχρι να γίνει πάλι η επαλήθευση τους [6].

- **Διαδικασίες (*Processes*):** Συνιστούν μια ακολουθία ενεργειών, οι οποίες αφού ενεργοποιηθούν, παραμένουν ενεργές μέχρι τον τερματισμό τους [6].
- **Συνθήκες (*Conditions*):** Χρησιμοποιούνται για να καθορίσουν τις προϋποθέσεις υπό τις οποίες λαμβάνει χώρα ένα γεγονός ή εκκινούν ή τερματίζουν μια διαδικασία [6].

Ο συνδυασμός όλων αυτών των στοιχείων επιτρέπει τη δημιουργία πολύπλοκων συστημάτων, με χαρακτηριστικότερο παράδειγμα την **εσωτερική οικονομία (internal economy)** του παιχνιδιού, ένα σύστημα όπου πόροι και οντότητες παράγονται, καταναλώνονται και ανταλλάσσονται σε υπολογίσιμες ποσότητες. Η ροή αυτή ελέγχεται από συγκεκριμένους μηχανισμούς: τις **πηγές (*sources*)**, που εισάγουν πόρους στο παιχνίδι, και τους **στραγγιστές (drains)**, που τους αφαιρούν οριστικά [6].

2.3.3. Προσέγγιση Σειριακού (Turn-Based) και Πραγματικού (Real Time) Χρόνου

Η φύση των βασικών μηχανισμών ενός παιχνιδιού διαφοροποιείται ριζικά ανάλογα με το αν αυτό λειτουργεί σε **πραγματικό χρόνο (*real-time*)** ή ακολουθεί **σειριακή δομή (*turn-based*)** [6].

Στα παιχνίδια πραγματικού χρόνου, ο κόσμος τους εξελίσσεται αυτόνομα. Συνεπώς, οι μηχανισμοί λειτουργούν κυρίως ως συνεχείς διεργασίες που εκτελούνται στο παρασκήνιο. Ωστόσο, ορισμένοι μηχανισμοί μπορεί να ενεργοποιούνται ως μεμονωμένα γεγονότα ως άμεση απάντηση σε μια ενέργεια του παίκτη ή στην εκπλήρωση μιας συνθήκης [6].

Αντιθέτως, στα σειριακά παιχνίδια, οι μηχανισμοί είναι αντιδραστικοί. Παραμένουν αδρανείς μέχρι ο παίκτης να ολοκληρώσει τη σειρά του και ενεργοποιούνται στιγμιαία για να υπολογίσουν και να εφαρμόσουν τις συνέπειες των πράξεων του, πριν επιστρέψουν ξανά σε κατάσταση αναμονής [6].

2.4. Μηχανή Παιχνιδιού (Game Engine)

Η **μηχανή του παιχνιδιού (*game engine*)** είναι το λογισμικό που υλοποιεί τους κανόνες του παιχνιδιού. Οι βασικοί μηχανισμοί περιγράφουν λεπτομερώς τους κανόνες του παιχνιδιού, καθορίζοντας τη συμπεριφορά της μηχανής, αλλά δεν επιβάλλουν το τρόπο υλοποίηση τους, αυτό αποτελεί αρμοδιότητα των προγραμματιστών [6].

2.4.1. Godot

Η *Godot* είναι μια μηχανή παιχνιδιών ανοιχτού λογισμικού (*open-source*) που περιέχει ένα ολοκληρωμένο περιβάλλον για την ανάπτυξη διοδιάστατων (*2D*) και τρισδιάστατων (*3D*) παιχνιδιών. Η μηχανή ενσωματώνει ένα σύνολο εργαλείων, όπως επεξεργαστή κώδικα (*code editor*) και πρόγραμμα εντοπισμού σφαλμάτων (*debugger*), τα οποία διευκολύνουν τη διαδικασία της ανάπτυξης. Ο κώδικας στην *Godot* μπορεί να γραφεί είτε σε *GDScript*, μια γλώσσα προγραμματισμού βελτιστοποιημένη για την μηχανή, είτε σε γλώσσα C#. Η λειτουργία της μηχανής θεμελιώνεται στις αρχές του αντικειμενοστραφούς προγραμματισμού [11].

Κύριες έννοιες της μηχανής Godot συνιστούν:

- **Σκηνές (*Scenes*):** Οι σκηνές αποτελούν τις βασικές δομικές μονάδες ενός παιχνιδιού στην *Godot*. Μια σκηνή μπορεί να αναπαριστά έναν χαρακτήρα, ένα στοιχείο του UI , ή ακόμα και ένα ολόκληρο επίπεδο. Λειτουργικά, οι σκηνές στην *Godot* επιτελούν τόσο το ρόλο των “σκηνών” όσο και των “prefabs” που συναντώνται σε άλλες μηχανές [11].

- **Κόμβοι (*Nodes*):** Οι κόμβοι είναι τα θεμελιώδη δομικά στοιχεία της μηχανής. Κάθε κόμβος είναι ένα αντικείμενο που διαθέτει συγκεκριμένες ιδιότητες και μπορεί να ενσωματωθεί σε μια ιεραρχική δομή δένδρου. Μια σκηνή απαρτίζεται από έναν ή περισσότερους κόμβους οργανωμένους σε δενδρική δομή [11].
- **Δένδρο Σκηνής (*Scene Tree*):** Το δένδρο σκηνής είναι η ενεργή ιεραρχία κόμβων κατά τον χρόνο εκτέλεσης του παιχνιδιού [11].
- **Σήματα (*Signals*):** Οι κόμβοι εκπέμπουν σήματα όταν συμβαίνει ένα συγκεκριμένο γεγονός. Άλλοι κόμβοι μπορούν να λαμβάνουν αυτά τα σήματα και να αντιδρούν ανάλογα. Το σύστημα αυτό, επιτρέπει την επικοινωνία μεταξύ των αντικειμένων, επιτρέποντας τη συγγραφή πιο ευέλικτου και συντηρήσιμου κώδικα [11].

2.5. Αρχές Εξισορρόπησης του Παιχνιδιού

Με την οριστικοποίηση του σχεδιασμού του παιχνιδιού, η διαδικασία εισέρχεται στο στάδιο της βελτιστοποίησης. Ο σχεδιαστής καλείται να διασφαλίσει πως το τελικό προϊόν που θα παράξει θα είναι δίκαιο και ελκυστικό για τον παίκτη. Ο *Adams* εισάγει ένα σύνολο εννοιών που ο σχεδιαστής πρέπει να υιοθετεί και να αποφεύγει για την επίτευξη αυτού του στόχου.

2.5.1. Κυρίαρχες Στρατηγικές (Dominant Strategies)

Μια **στρατηγική (*strategy*)** ορίζεται ως η προσέγγιση που υιοθετεί ένας παίκτης εντός του παιχνιδιού με σκοπό την επίτευξη της νίκης. Μια **κυρίαρχη στρατηγική (*dominant strategy*)** αναφέρεται σε μια στρατηγική που παράγει με συνέπεια το βέλτιστο δυνατό αποτέλεσμα για τον παίκτη, ανεξαρτήτως των επιλογών του αντιπάλου. Η ύπαρξη κυρίαρχων στρατηγικών σε ένα παιχνίδι θεωρείται ανεπιθύμητη, καθώς αχρηστεύουν όλες τις άλλες επιλογές του παίκτη, περιορίζοντας το στρατηγικό ενδιαφέρον της εμπειρίας. Ιδιαίτερα προβληματική είναι η περίπτωση όπου μια κυρίαρχη στρατηγική είναι διαθέσιμη σε έναν παίκτη αλλά όχι στον αντίπαλο του, καθώς έτσι το παιχνίδι καθίσταται άδικο. Συνεπώς, ο σχεδιαστής οφείλει να εξαλείφει τις κυρίαρχες στρατηγικές για να διασφαλίσει την ισορροπία στο παιχνίδι του [6].

2.5.2. Μεταβατικές (Transitive) και Μη-Μεταβατικές (Intransitive) Σχέσεις

Για την αποτροπή των κυρίαρχων στρατηγικών, ο σχεδιαστής είναι απαραίτητο να είναι σε θέση να αναγνωρίζει τις **μεταβατικές σχέσεις (*transitive relationships*)** μεταξύ των επιλογών του παίκτη και να τις εξαλείφει. Ο όρος “μεταβατική” προσδιορίζει μια σχέση μεταξύ τριών ή περισσότερων οντοτήτων , σύμφωνα με την οποία εάν η οντότητα Α σχετίζεται με την Β και η Β σχετίζεται με την Γ, τότε η Α σχετίζεται με τον ίδιο τρόπο και με την Γ [6].

Όταν υπάρχει μια μεταβατική σχέση υπεροχής, όπου η επιλογή Α είναι ανώτερη της Β και η Β είναι ανώτερη της Γ, τότε η επιλογή Γ γίνεται μη βιώσιμη. Για την εξισορρόπηση του παιχνιδιού όταν υπάρχουν τέτοιες σχέσεις , ο σχεδιαστής μπορεί να εισάγει ένα **κόστος (*cost*)** στην ανώτερη επιλογή, παρέχοντας κίνητρο στο παίκτη για την επιλογή της λιγότερο αποτελεσματικής, αλλά ενδεχομένως οικονομικότερης εναλλακτικής [6].

Αντιθέτως **οι μη-μεταβατικές σχέσεις (*intransitive relationships*)** μεταξύ των επιλογών του παίκτη αποτελούν έναν αποτελεσματικό εργαλείο για την διασφάλιση της ισορροπίας του παιχνιδιού. Σε αυτό το είδος σχέσης, εάν η επιλογή Α είναι ανώτερη της Β και η Β ανώτερη της Γ, αυτό δεν συνεπάγει ότι η Α είναι ανώτερη της Γ. Το πιο

χαρακτηριστικό παράδειγμα μη-μεταβατικής σχέσης εντοπίζεται στο παιχνίδι "Πέτρα-Ψαλίδι-Χαρτί", όπου κάθε επιλογή είναι ανώτερη από μια άλλη και κατώτερη από μια τρίτη. Κατά τον σχεδιασμό των επιλογών του παίκτη, είναι επιθυμητό οι μεταξύ τους μη-μεταβατικές σχέσεις να είναι πιο σύνθετες και λιγότερο προφανείς, ώστε ο παίκτης να ωθείται στην εξερεύνηση διαφορετικών στρατηγικών [6].

2.5.3. Ορθογώνια Διαφοροποίηση Μονάδων (Orthogonal Unit Differentiation)

Για τη διασφάλιση του στρατηγικού βάθους και την αντιμετώπιση των κυρίαρχων στρατηγικών, εισάγεται και η έννοιά της **"ορθογώνιας διαφοροποίησης μονάδων" (*orthogonal unit differentiation*)**, όπως αυτή διατυπώθηκε από τον *Harvey Smith* στο *Game Developers Conference* του 2003. Με τον όρο "ορθογώνια", ο *Smith* τόνισε ότι οι μονάδες του παιχνιδιού θα πρέπει να διαφοροποιούνται σε πολλαπλές, διακριτές διαστάσεις και όχι απλώς να ιεραρχούνται σε μια μοναδική κλίμακα ισχύος [6].

Κατ' αυτόν τον τρόπο, οι επιλογές του παίκτη γίνονται πιο ενδιαφέρουσες, διότι κάθε μονάδα διαθέτει μοναδικά πλεονεκτήματα και καμία μεμονωμένη μονάδα δεν αποδεικνύεται καθολικά ανώτερη. Χαρακτηριστικό παράδειγμα αυτής της αρχής αποτελεί το σκάκι. Η βασίλισσα μπορεί να είναι το ισχυρότερο κομμάτι στο ταμπλό, μα μια στρατηγική που επικεντρώνεται αποκλειστικά στην αξιοποίηση της δεν ισοδυναμεί με κυρίαρχη στρατηγική, διότι και τα υπόλοιπα κομμάτια επιτελούν ένα διακριτό ρόλο, γεγονός που οδηγεί τον παίκτη στον συνδυασμό τους για την επίτευξη της νίκης [6].

2.6. Αλγόριθμοι Εξισορρόπησης του Παιχνιδιού

Οι διεργασίες των μηχανισμών αξιοποιούν κυρίως τις σχέσεις και τις τιμές των οντοτήτων, με σκοπό να προσδιορίσουν και να εφαρμόσουν τις απαραίτητες μεταβολές στο περιβάλλον του παιχνιδιού. Οι μηχανισμοί αυτοί συνήθως υπολογίζουν τις τιμές των παραμέτρων που θα επιφέρουν αλλαγές το κόσμο του παιχνιδιού. Ωστόσο, πολλές τιμές, όπως τα στατιστικά των χαρακτήρων, αποτελούν στατικές παραμέτρους, οι οποίες οφείλουν να είναι προκαθορισμένες πριν την έναρξη του παιχνιδιού [6].

Κατά το στάδιο της βελτιστοποίησης, ο σχεδιαστής καλείται να προβεί στη διαμόρφωση αυτών των τιμών, ώστε να προκύψει ένα ισορροπημένο αποτέλεσμα, παρέχοντας ταυτόχρονα το επιθυμητό επίπεδο πρόκλησης στο παίκτη. Η διαδικασία αυτή για τους σκοπούς αυτής της εργασίας θα πραγματοποιηθεί μέσω της εφαρμογής ενός συγκεκριμένου συνόλου αλγορίθμων [6].

2.6.1. Εξαντλητική Αναζήτηση (Exhaustive Search)

Η **εξαντλητική αναζήτηση (*exhaustive search*)** είναι μια αλγοριθμική προσέγγιση που ανήκει στην κατηγορία αλγορίθμων **"ωμής βίας" (brute force)** και εφαρμόζεται για την επίλυση συνδυαστικών προβλημάτων. Η μεθοδολογία της συνίσταται σε τρία διακριτά στάδια:

1. Την παραγωγή του πλήρους **χώρου λύσεων (*solution space*)** του προβλήματος [12].
2. Τον έλεγχο κάθε υποψήφιας λύσης ως προς την τήρηση των δεδομένων περιορισμών [12].
3. Τον εντοπισμό της λύσης εκείνης που βελτιστοποιεί μια προκαθορισμένη **αντικειμενική συνάρτηση (*objective function*)** [12].

2.6.2. Προβλήματα Ικανοποίησης Περιορισμών (Constraint Satisfaction Problems)

Ένα **πρόβλημα ικανοποίησης περιορισμών (*constraint satisfaction problem*)** αποτελείται από μια τριάδα συνόλων, τις **μεταβλητές (*variables*)**, τα **πεδία (*domains*)** και τους **περιορισμούς (*constraints*)** [13].

- Ένα πεδίο, ορίζεται ως το σύνολο των **επιτρεπόμενων τιμών (*values*)** που μπορεί να λάβει μια μεταβλητή. Διαφορετικές μεταβλητές μπορούν να έχουν πεδία διαφορετικού μεγέθους [13].
- Κάθε περιορισμός, αποτελείται από ένα ζεύγος **<εμβέλεια, σχέση>**, όπου η **εμβέλεια (*scope*)** είναι μια πλειάδα που περιλαμβάνει τις μεταβλητές που συμμετέχουν στον αντίστοιχο περιορισμό, και η **σχέση (*relationship*)** προσδιορίζει τους επιτρεπτούς συνδυασμούς τιμών για αυτές τις μεταβλητές. Μια σχέση μπορεί να αναπαρίσταται ρητά ως ένα σύνολο όλων των πλειάδων που ικανοποιούν τον περιορισμό ή ως μια συνάρτηση η οποία επαληθεύει εάν μια δεδομένη πλειάδα είναι έγκυρη [13].

Η κύρια λειτουργία των αλγορίθμων επίλυσης προβλημάτων ικανοποίησης περιορισμών είναι η ανάθεση τιμών στο σύνολο των μεταβλητών [13].

- Μια ανάθεση χαρακτηρίζεται ως **συνεπής (*consistent assignment*)** εφόσον δεν παραβιάζει κανέναν περιορισμό [13].
- Μια ανάθεση ορίζεται ως **πλήρης (*complete assignment*)** αν συμπληρώνει μια τιμή σε κάθε μεταβλητή του συνόλου [13].
- Μια ανάθεση θεωρείται **μερική (*partial assignment*)** εφόσον δεν αναθέτει όλες τις μεταβλητές του συνόλου [13].

Τέλος, ως **λύση (*solution*)** ενός προβλήματος ικανοποίησης περιορισμών ορίζεται μια ανάθεση η οποία είναι συνεπής και πλήρης, ενώ ως **μερική λύση (*partial solution*)** θεωρείται μια ανάθεση που είναι συνεπής και μερική [13].

2.6.3. Προσομοίωση (Simulation)

Ως **προσομοίωση (*simulation*)** ορίζεται η αναπαράσταση της λειτουργίας ενός συστήματος με την πάροδο του χρόνου, δημιουργώντας έτσι μια **τεχνητή ιστορία (*artificial history*)** του. Η παρατήρηση αυτής της ιστορίας αποσκοπεί στην εξαγωγή συμπερασμάτων αναφορικά με τα λειτουργικά χαρακτηριστικά του πραγματικού συστήματος [14].

Μέσω της ανάπτυξης ενός **μοντέλου προσομοίωσης (*simulation model*)**, καθίσταται δυνατή η μελέτη της συμπεριφοράς του συστήματος. Το μοντέλο αυτό συνήθως λαμβάνει τη μορφή ενός συνόλου υποθέσεων αναφορικά με τη συμπεριφορά του συστήματος, οι οποίες εκφράζονται μέσω μαθηματικών, λογικών ή συμβολικών σχέσεων μεταξύ των οντοτήτων του. Με την ανάπτυξη και την επικύρωση του μοντέλου, αυτό μπορεί να χρησιμοποιηθεί για τη διερεύνηση υποθετικών σεναρίων σχετικά με το πραγματικό σύστημα [14].

Για την πληρέστερη ανάλυση και κατανόηση ενός συστήματος ορίζονται οι εξής έννοιες:

- Μια **οντότητα (*entity*)** είναι ένα αντικείμενο ενδιαφέροντος εντός του συστήματος [14].
- Ένα **χαρακτηριστικό (*attribute*)** είναι μια ιδιότητα μιας οντότητας [14].
- Η **κατάσταση (*state*)** του συστήματος ορίζεται ως το σύνολο των μεταβλητών που απαιτούνται για τον προσδιορισμό του συστήματος σε οποιαδήποτε δεδομένη χρονική στιγμή, αναφορικά με τους στόχους της ανάλυσης [14].

Ένα σύστημα είναι **διακριτό (*discrete*)** όταν οι μεταβλητές της κατάστασης του μεταβάλλονται μόνο σε διακριτές χρονικές στιγμές, αντιθέτως ορίζεται ως **συνεχές (*continuous*)** όταν αλλάζουν συνεχώς με την πάροδο του χρόνου [14].

Τα μοντέλα προσομοίωσης μπορούν να διαχωριστούν στις παρακάτω κατηγορίες:

- Ένα **στατικό (*static*)** μοντέλο αναπαριστά ένα σύστημα σε μια συγκεκριμένη χρονική στιγμή, ενώ ένα **δυναμικό (*dynamic*)** καθώς αλλάζει με την πάροδο του χρόνου [14].
- Τα μοντέλα προσομοίωσης που δεν περιέχουν τυχαίες μεταβλητές χαρακτηρίζονται ως **ντετερμινιστικά (*deterministic*)**. Τα μοντέλα αυτά έχουν γνωστές εισόδους και παράγουν μοναδικά σύνολα αποτελεσμάτων. Αντιθέτως, τα **στοχαστικά (*stochastic*)** μοντέλα προσομοίωσης εμπεριέχουν τουλάχιστον μια τυχαία μεταβλητή, η οποία παράγει τυχαίο αποτέλεσμα. Τα αποτελέσματα αυτά, λόγω της τυχαίας φύσης τους, μπορούν να θεωρηθούν ως εκτιμήσεις των πραγματικών χαρακτηριστικών ενός μοντέλου [14].

2.6.4. Μέτρα Τοποθεσίας (Measures of Location)

Τα **μέτρα τοποθεσίας (*measures of location*)** προσδιορίζουν ποσοτικά την **κεντρική τάση (*central tendency*)** ή άλλη χαρακτηριστική θέση ενός συνόλου δεδομένων [15].

Το **δειγματικό μέσο (*sample mean*)**, το οποίο συμβολίζεται με $\bar{x}$, ορίζεται ως ο αριθμητικός μέσος όρος των τιμών ενός δείγματος μεγέθους n, και υπολογίζεται από τον τύπο:

$$\bar{x} = \sum_{i=1}^{n} \frac{x_i}{n} = \frac{x_1 + x_2 + .. + x_n}{n} \quad (2.1) \quad [15]$$

Η **δειγματική διάμεσος (*sample median*)** αποτελεί ένα μέτρο κεντρικής τάσης το οποίο είναι ανθεκτικό στην παρουσία ακραίων τιμών (outliers). Συμβολίζεται με $\tilde{x}$ και υπολογίζεται ως εξής:

$$\tilde{x} = \begin{cases} x_{(n+1)/2}, & \text{αν το } n \text{ είναι περιττός} \\ \frac{1}{2}(x_{n/2} + x_{n/2+1}), & \text{αν το } n \text{ είναι άρτιος} \end{cases} \quad (2.2) \quad [15]$$

2.7. Αλγόριθμος Τεχνητής Νοημοσύνης

Όπως θα αναλυθεί στη συνέχεια, το σύστημα μάχης που θα υλοποιηθεί για τους σκοπούς του παιχνιδιού της παρούσας εργασίας ,επικεντρώνεται σε τρεις διακριτούς χαρακτήρες. Οι ενέργειες αυτών των τριών χαρακτήρων καθορίζουν το τελικό αποτέλεσμα του παιχνιδιού. Για τους δύο από τους τρεις χαρακτήρες, η επιλογή των ενεργειών τους καθορίζεται απευθείας από τους μηχανισμούς του παιχνιδιού. Αντιθέτως, ο τρίτος χαρακτήρας απαιτεί την εφαρμογή ενός εξειδικευμένου αλγορίθμου για την επιλογή των κινήσεων του. Ο βασικός στόχος του αλγορίθμου αυτού είναι η πρόβλεψη των μελλοντικών καταστάσεων του παιχνιδιών ,επιτρέποντας την επιλογή της ενέργειας που θα επιφέρει το βέλτιστο δυνατό αποτέλεσμα. Η επίτευξη του στόχου αυτού καθίσταται δυνατή μέσω της αξιοποίησης ενός συνόλου αλγορίθμων.

2.7.1. Προβλήματα Αναζήτησης

Ένα πρόβλημα αναζήτησης ορίζεται από τις παρακάτω έννοιες:

- **Χώρος Καταστάσεων(*Space States*):** Είναι το σύνολο των πιθανών **καταστάσεων (*states*)** στις οποίες μπορεί να βρίσκεται το περιβάλλον του

προβλήματος. Αυτός ο χώρος μπορεί να αναπαρασταθεί από έναν γράφο, όπου οι κορυφές είναι οι καταστάσεις και οι κατευθυνόμενες ακμές οι ενέργειες [13].

- **Αρχική Κατάσταση (*Initial State*):** Είναι η κατάσταση στην οποία ξεκινάει η αναζήτηση της λύσης του προβλήματος [13].
- **Καταστάσεις-Στόχος (*Goal States*):** Είναι ένα σύνολο μιας ή περισσότερων καταστάσεων, η επίτευξη των οποίων συνιστά τη λύση του προβλήματος. Σε μερικά προβλήματα είναι πιθανό να υπάρχει μια κατάσταση-στόχος, ενώ σε άλλα ένα μικρό σύνολο με εναλλακτικές καταστάσεις-στόχους. Σε συγκεκριμένες περιπτώσεις, ο στόχος ορίζεται από μια ιδιότητα που εφαρμόζεται σε πολλές καταστάσεις [13].
- **Διαθέσιμες Ενέργειες (*Actions*):** Για μια δεδομένη κατάσταση s, μια συνάρτηση *ACTIONS(s)* επιστρέφει ένα πεπερασμένο σύνολο ενεργειών που μπορούν να εκτελεστούν στην κατάσταση s [13].
- **Μοντέλο Μετάβασης (*Transition Model*):** Περιγράφει το αποτέλεσμα που έχει η εφαρμογή κάθε ενέργειας. Συγκεκριμένα, μια συνάρτηση RESULT(s, a) επιστρέφει την κατάσταση-αποτέλεσμα της εφαρμογής της ενέργειας a στην κατάσταση s [13].
- **Συνάρτηση Κόστους Ενέργειας (*Action Cost Function*):** Μια συνάρτηση, που συμβολίζεται ως ACTION-COST(s, ,a, s'), αποδίδει το αριθμητικό κόστος για την εφαρμογή της ενέργειας a στην κατάσταση s, με σκοπό τη μετάβαση στην κατάσταση s' [13].
- **Μονοπάτι (*Path*):** Είναι μια ακολουθία ενεργειών [13].
- **Λύση (*Solution*):** Είναι το μονοπάτι από την αρχική κατάσταση στην κατάσταση-στόχο [13].

Η επίλυση ενός προβλήματος αναζήτησης επιτυγχάνεται μέσω της εφαρμογής ενός αλγορίθμου αναζήτησης [13].

2.7.2 Δεντρικές Δομές

Ένα δένδρο είναι μια συλλογή από στοιχεία που ονομάζονται **κόμβοι (*nodes*)**. Ένας από αυτούς τους κόμβους καθορίζεται ως η **ρίζα (*root*)** και μια **σχέση γονέα-παιδιού (*parenthood*)** παρέχει μια ιεραρχική δομή μεταξύ των κόμβων [16].

Ένα δένδρο μπορεί να οριστεί αναδρομικά με τον ακόλουθο τρόπο:

1. Ένας μεμονωμένος κόμβος αποτελεί ένα δένδρο. Αυτός ο κόμβος είναι η ρίζα του δένδρου [16].
2. Έστω n ένας κόμβος και $T_1, T_2, \dots, T_k$ ένα σύνολο από δένδρα με ρίζες $n_1, n_2, \dots, n_k$ αντίστοιχα. Ένα νέο δένδρο δημιουργείται αν ο κόμβος n γίνει ο γονέας των κόμβων $n_1, n_2, \dots, n_k$. Σε αυτό το δένδρο, ο κόμβος n είναι η ρίζα και τα $T_1, T_2, \dots, T_k$ είναι τα υπο-δένδρα της ρίζας. Οι κόμβοι $n_1, n_2, \dots, n_k$ ονομάζονται **παιδιά (*children*)** του n [16].

Εάν $n_1, n_2, \dots, n_k$ είναι μια ακολουθία από κόμβους σε ένα δένδρο όπου το n_i είναι ο γονέας του n_{i+1} για $i \leq i < k$, τότε η ακολουθία κόμβων από το n_1 στο n_k ονομάζεται **μονοπάτι (*path*)**. Το μήκος του μονοπατιού είναι κατά ένα μικρότερο από τον αριθμό των κόμβων του. Συνεπώς, ένα μονοπάτι με μήκος μηδέν αποτελείται από έναν μόνο κόμβο [16].

Εάν υπάρχει ένα μονοπάτι από έναν κόμβο α σε έναν κόμβο β, ο κόμβος α είναι **πρόγονος (*ancestor*)** του β και ο β **απόγονος (*descendant*)** του α. Ένας πρόγονος ή απόγονος ενός κόμβου, πέρα του εαυτού του, ονομάζεται **γνήσιος πρόγονος (*proper ancestor*)** ή **γνήσιος απόγονος (*proper descendant*)** αντίστοιχα. Σε ένα δένδρο, η ρίζα είναι ο μόνος κόμβος χωρίς γνήσιο πρόγονο και τα **φύλλα (*leaves*)** είναι οι μόνοι κόμβοι χωρίς γνήσιο απόγονο [16].

Το **ύψος (*height*)** ενός κόμβου σε ένα δένδρο είναι το μήκος του μακρύτερου μονοπατιού από αυτό σε ένα φύλλο. Το ύψος ενός δένδρου είναι το ύψος της ρίζας. Το **βάθος (*depth*)** ενός κόμβου είναι το μήκος του μονοπατιού που οδηγεί από την ρίζα σε αυτό [16].

2.7.3. Αλγόριθμοι Minimax και Expectiminimax

Ένα **δένδρο παιχνιδιού (*game tree*)** είναι μια ρητή αναπαράσταση όλων των δυνατών **παρτίδων (*plays*)** ενός παιχνιδιού. Ο κόμβος της ρίζας είναι η αρχική κατάσταση του παιχνιδιού, οι διάδοχοι του αναπαριστούν τις καταστάσεις που προκύπτουν από τις κινήσεις του πρώτου παίκτη, οι διάδοχοι αυτών των κόμβων τις καταστάσεις που προκύπτουν από τις κινήσεις του δεύτερου, και ούτω καθεξής. Τα φύλλα του δένδρου αναπαριστούν νίκη, ήττα ή ισοπαλία. Κάθε μονοπάτι από την ρίζα σε ένα φύλλο αναπαριστά μια διαφορετική πλήρη παρτίδα του παιχνιδιού [17].

Στο παιχνίδι το οποίο υλοποιείται το δένδρο, ο πρώτος παίκτης ονομάζεται *MAX* και ο αντίπαλος του *MIN*. Το δένδρο αυτό, περιλαμβάνει δύο είδη κόμβων, τους *MAX* κόμβους (σε άρτια επίπεδα από την ρίζα) και *MIN* (σε περιττά επίπεδα από την ρίζα) που χαρακτηρίζονται από το όνομα του παίκτη στον οποίον αντιστοιχούν. Οι *MAX* κόμβοι αναπαρίστανται με τετράγωνα και οι *MIN* κόμβοι με κύκλους. Τα φύλλα έχουν τις ετικέτες *WIN*, *LOSS*, *DRAW* με βάση το αποτέλεσμα του παιχνιδιού από την προοπτική του *MAX* [17].

Αφού οριστούν οι ετικέτες *WIN*, *LOSS*, *DRAW* στα φύλλα, κάθε κόμβος στο παιχνίδι μπορεί να έχει μια ετικέτα *WIN*, *LOSS* ή *DRAW* που προκύπτει από την παρακάτω διαδικασία:

1. Αν J είναι ένας μη τερματικός *MAX* κόμβος τότε:
   - ***STATUS(J)*** = {***WIN***, αν οποιοσδήποτε διάδοχος του είναι *WIN*. ***LOSS***, αν όλοι οι διάδοχοι του είναι *LOSS*. ***DRAW***, αν οποιοσδήποτε διάδοχος του είναι *DRAW* και κανένας δεν είναι *WIN*} [17].
2. Αντίστοιχα, αν J είναι ένας μη τερματικός MIN κόμβος τότε:
   - ***STATUS(J)*** = {***WIN***, αν όλοι οι διάδοχοι είναι *WIN*. ***LOSS***, αν ένας διάδοχος του είναι *LOSS*. ***DRAW***, αν οποιοσδήποτε διάδοχος του είναι *DRAW* και κανένας δεν είναι *LOSS*} [17].

Η συνάρτηση *STATUS(J)* αναπαριστά το καλύτερο τερματικό αποτέλεσμα που μπορεί να επιτύχει ο παίκτης *MAX* αν παίζει βέλτιστα από την κατάσταση J εναντίον ενός τέλειου αντιπάλου [17].

Στα περισσότερα παιχνίδια όμως οι πιθανότητες είναι πάρα πολλές για την δημιουργία όλων των τερματικών κόμβων και την αξιολόγηση τους προς τα πίσω ώστε να προκύψει η πρώτη βέλτιστη κίνηση. Η έλλειψη πρακτικών τρόπων αξιολόγησης του αποτελέσματος των διαδοχικών καταστάσεων του παιχνιδιού, οδηγεί σε **ευρετικές προσεγγίσεις (*heuristic approximations*)**. Εμπειρικά, είναι προφανές ότι ορισμένα στοιχεία σε μια κατάσταση του παιχνιδιού συνεισφέρουν στην ισχύ της, ενώ αλλά στην αδυναμία της. Η λύση είναι δημιουργία μιας **συνάρτησης αξιολόγησης (*evaluation function*) e**, η οποία συνδυάζει αυτά τα στοιχεία για να υπολογίσει μια αριθμητική "αξία" για κάθε κατάσταση. Στη συνέχεια, ο αλγόριθμος επιλέγει την κίνηση που οδηγεί στην κατάσταση με την υψηλότερη εκτιμώμενη αξία [17].

Κανόνας Minimax: Οι κόμβοι του δένδρου παιχνιδιού αξιολογούνται με τον παρακάτω κανόνα:

- Η τιμή *V(J)* ενός κόμβου J που βρίσκεται στο **μέτωπο της αναζήτησης (*search frontier*)** είναι ίση με την τιμή της στατικής συνάρτησης αξιολόγησης e(J) [13].
- Η τιμή *V(J)* ενός εσωτερικού κόμβου (J) *MAX* είναι ίση με την μέγιστη τιμή από τις τιμές των διαδόχων του [13].

- Η τιμή *V(J)* ενός εσωτερικού κόμβου (J) *MIN* είναι ίση με την μέγιστη τιμή από τις τιμές των διαδόχων του [13].

Όταν η στρατηγική του αντιπάλου δεν είναι βέλτιστα ανταγωνιστική αλλά περιλαμβάνει ένα στοιχείο τυχαιότητας, το πρόβλημα παύει να είναι ντετερμινιστικό και μετατρέπεται σε **στοχαστικό (*stochastic game*)**. Η μοντελοποίηση της τυχαιότητας απαιτεί την εισαγωγή ενός νέου είδους κόμβων στο δένδρο του παιχνιδιού, των **κόμβων τύχης (*chance nodes*)** [13].

Αυτό οδηγεί στον **αλγόριθμο *Expectiminimax***, ο οποίος αποτελεί γενίκευση του ***Minimax***. Λόγω της τυχαιότητας, η αξία μιας κατάστασης παύει να είναι ντετερμινιστική. Για τον λόγο αυτό, η μόνη λύση είναι ο υπολογισμός της **αναμενόμενης τιμής (*expected value*)** της κατάστασης, που είναι ο μέσος όρων των πιθανών αποτελεσμάτων της. Ενώ οι κόμβοι *MAX* και *MIN* λειτουργούν με τον ίδιο τρόπο όπως στον *Minimax* αλγόριθμο, για τους κόμβους τύχης υπολογίζεται η αναμενόμενη τιμή, η οποία είναι το σταθμισμένο άθροισμα των τιμών όλων πιθανών εκβάσεων, όπου κάθε τιμή πολλαπλασιάζεται με την πιθανότητα της να συμβεί [13].

Κεφάλαιο 3- Σχεδιασμός και Υλοποίηση του Ηλεκτρονικού Παιχνιδιού

Στο παρόν κεφάλαιο, θα αξιοποιηθούν οι εννοιολογικές αρχές που συστήθηκαν στο κεφάλαιο 2 "***Θεωρητικό Πλαίσιο***", με σκοπό να προσεγγιστεί μεθοδικά ο σχεδιασμός και η ανάπτυξη ενός ηλεκτρονικού παιχνιδιού.

Οι *ενότητες 3.1* και *3.2*, εστιάζουν στο "*στάδιο της ιδέας*". Ειδικότερα, στην πρώτη ενότητα θα καθοριστούν οι θεμελιώδεις αποφάσεις που θα διαμορφώσουν το παιχνίδι, ενώ στη δεύτερη θα τεθούν οι γενικοί κανόνες λειτουργίας του.

Οι *ενότητες 3.3* έως *3.5* αφορούν το "*στάδιο της επεξεργασίας*". Αρχικά, στην *ενότητα 3.3*. θα διαμορφωθούν τα δομικά στοιχεία του σχεδιασμού. Στην συνέχεια, στην *ενότητα 3.4*, αυτά τα δομικά στοιχεία, σε συνδυασμό με τους κανόνες του παιχνιδιού, θα αποτελέσουν τη βάση για τον αναλυτικό σχεδιασμό των μηχανισμών του. Τέλος, στην *ενότητα 3.5* θα πραγματοποιηθεί η προγραμματιστική υλοποίηση αυτών των μηχανισμών.

Η *ενότητα 3.6* επικεντρώνεται στο "*στάδιο της βελτιστοποίησης*". Αρχικά θα προσδιοριστούν τα στατικά δεδομένα του παιχνιδιού, τα οποία μπορούν να οριστούν βάσει των αρχών βελτιστοποίησης που παρατέθηκαν στην ενότητα *2.4 "**Αρχές Εξισορρόπησης του Παιχνιδιού**"*. Ακολούθως, μέσω της εφαρμογής ενός συνόλου αλγορίθμων, θα καθοριστούν τα εναπομείναντα στατικά δεδομένα του παιχνιδιού, με σκοπό τη διασφάλιση μιας δίκαιης και ισορροπημένης εμπειρίας για τον παίκτη.

3.1. Θεμελιώδεις Αποφάσεις

Η πρώτη θεμελιώδης σχεδιαστική απόφαση κατά τη διαδικασία ανάπτυξης ενός ηλεκτρονικού παιχνιδιού είναι ο καθορισμός της κεντρικής ιδέας. Ο καθορισμός αυτής της ιδέας διευκολύνεται από τον προσδιορισμό του είδους στο οποίο ο σχεδιαστής σκοπεύει να εντάξει το παιχνίδι. Το παρόν έργο εντάσσεται στο είδος των **παιχνιδιών ρόλων (*Role-Playing Games- RPGs*)**. Συνεπώς, το παιχνίδι σχεδιάζεται ώστε να ενσωματώνει τα βασικά χαρακτηριστικά των ηλεκτρονικών παιχνιδιών ρόλων, όπως αυτά προσδιορίστηκαν στην *ενότητα 1.4*.

Συγκεκριμένα, η κεντρική ιδέα του παιχνιδιού διατυπώνεται ως εξής: "Μια ομάδα που αποτελείται από έναν μαχητή και έναν μάγο ταξιδεύουν σε ένα μαγικό δάσος για να αντιμετωπίσουν υπερφυσικά πλάσματα ".

Έχοντας ορίσει την κεντρική ιδέα, το επόμενο στάδιο της μεθοδολογίας αφορά τον προσδιορισμό του κοινού-στόχου. Δεδομένου ότι το παιχνίδι αναπτύσσεται στο πλαίσιο της παρούσας πτυχιακής εργασίας και δεν προορίζεται για εμπορική κυκλοφορία, δεν είναι δυνατή η ύπαρξη πραγματικού κοινού. Ωστόσο, για τις ανάγκες της σχεδιαστικής διαδικασίας, θα θεωρηθεί ένα υποθετικό κοινό. Εφόσον το παιχνίδι ανήκει στο είδος των παιχνιδιών ρόλων, ως κοινό-στόχος επιλέγονται οι παίκτες που προτιμούν τη συγκεκριμένη κατηγορία παιχνιδιών. Η απόφαση αυτή επιτρέπει στον σχεδιασμό να εστιάσει στην ενσωμάτωση των θεμελιωδών χαρακτηριστικών που ορίζουν την ταυτότητα αυτών των παιχνιδιών.

Στο παρόν παιχνίδι, ο παίκτης θα ενσαρκώνει τον ρόλο ενός μαχητή. Το αρχέτυπο αυτό είναι κλασικό για το συγκεκριμένο είδος παιχνιδιών, με την παρουσία του να είναι ήδη καθιερωμένη από την πρώτη έκδοση του *Dungeons & Dragons***,** του πρώτου εμπορικά διαθέσιμου παιχνιδιού ρόλων. Οι παίκτες που επιλέγουν τον ρόλο του μαχητή σε ηλεκτρονικά παιχνίδια ρόλων, τυπικά αναμένουν να έχουν στη διάθεση τους ένα ευρύ φάσμα επιλογών τόσο στον οπλισμό όσο και στις επιθετικές ενέργειες τους. Αντίστοιχα, όσον αφορά τις προκλήσεις, είναι σύνηθες σε τέτοιου είδους παιχνίδια η αντιμετώπιση εχθρών με ξεχωριστές ικανότητες, απαιτώντας την ανάπτυξη ευέλικτων στρατηγικών. Συνεπώς, κατά τον σχεδιασμό του συστήματος μάχης, είναι απαραίτητο να δοθεί έμφαση στην ποικιλία των επιλογών τόσο για τον παίκτη όσο και για τους αντιπάλους. Ο σχεδιασμός αυτός αποσκοπεί στην παρακίνηση του παίκτη για πειραματισμό και συνδυασμό των διαθέσιμων ενεργειών, με στόχο την επίτευξη της νίκης.

Με τις δημιουργικές αποφάσεις του παιχνιδιού να είναι καθορισμένες, η διαδικασία μπορεί να προχωρήσει στον προσδιορισμό του εύρους. Μια βέλτιστη μεθοδολογία για την εκτίμηση του εύρους αποτελεί η διερεύνηση ενός ανάλογου παιχνιδιού και η σύγκριση των απαιτούμενων πόρων για την ανάπτυξη του σε σχέση με τους διαθέσιμους πόρους της παρούσας εργασίας. Ένα από τα πλέον αναγνωρισμένα παραδείγματα ηλεκτρονικού παιχνιδιού ρόλων που αναπτύχθηκε από μια μικρή ομάδα με περιορισμένο προϋπολογισμό είναι το *Undertale* (2015) του *Toby Fox*. Συγκεκριμένα, για το *Undertale:*

- Η ανάπτυξη του πραγματοποιήθηκε κυρίως από τον δημιουργό του, *Toby Fox*, ο οποίος ανέλαβε το μεγαλύτερο εργασιακό φόρτο, με την *Temmie Chang* να συνεισφέρει στον τομέα του σχεδιασμού γραφικών [18].
- Ο προϋπολογισμός του ανήλθε στα 51.000 δολάρια, ποσό το οποίο συγκεντρώθηκε μέσω της ηλεκτρονικής πλατφόρμας *Kickstarter* [18].
- Η περίοδος ανάπτυξης του διήρκησε 2 έτη και 7 μήνες [18].

Λαμβάνοντας υπόψη τους πόρους που χρειάστηκαν για την ανάπτυξη του *Undertale* καθίσταται σαφές ότι η δημιουργία ενός πλήρους ηλεκτρονικού παιχνιδιού ρόλων στο πλαίσιο της πτυχιακής εργασίας είναι απαγορευτική. Παρότι ο προϋπολογισμός του *Undertale* είναι συγκριτικά μικρός για τα δεδομένα της βιομηχανίας, παραμένει ένα ποσό απαγορευτικό για τις δυνατότητες της συγκεκριμένης εργασίας. Πέραν του οικονομικού περιορισμού, ο απαιτούμενος χρόνος ανάπτυξης, που στην περίπτωση του *Undertale* ήταν 2 χρόνια και 7 μήνες, συνιστά ένα εξίσου σημαντικό περιορισμό, καθώς υπερβαίνει το διαθέσιμο χρονοδιάγραμμα που έχει τεθεί για την εργασία. Συνεπώς, εφόσον η ανάπτυξη ενός πλήρους παιχνιδιού ρόλων υπερβαίνει το καθορισμένο εύρος του έργου, κρίνεται αναγκαία μια τροποποίηση της σχεδιαστικής προσέγγισης. Η τροποποίηση αυτή είναι η μεταβολή της μορφής του έργου, η οποία αντί για ένα ολοκληρωμένο παιχνίδι, θα είναι ένα λειτουργικό πρωτότυπο το οποίο θα επικεντρώνεται αποκλειστικά στο σύστημα μάχης. Αυτή η προσέγγιση επιτρέπει την εφαρμογή των σχεδιαστικών αρχών του *Adams* και τη δημιουργία ενός αξιολογήσιμου τελικού προϊόντος.

3.2. Κανόνες του Παιχνιδιού

Με τα βασικά στοιχεία του παιχνιδιού οριστικοποιημένα, ακολουθεί ο προσδιορισμός των κανόνων του, οι οποίοι στη συνέχεια θα μετατραπούν σε μηχανισμούς που θα ρυθμίζουν τη λειτουργία του συστήματος μάχης του πρωτοτύπου.

Αρχικά, ορίζονται οι τρεις συμμετέχοντες χαρακτήρες του παιχνιδιού: ο Παίκτης (*Player*), ο οποίος ελέγχεται από τον χρήστη, ο Βοηθός (*Helper*), που ελέγχεται από ένα σύστημα Τεχνητής Νοημοσύνης (*AI*) και ο Αντίπαλος (*Enemy*), του οποίου οι ενέργειες καθορίζονται από έναν αλγόριθμο παραγωγής τυχαίων αριθμών. Κάθε χαρακτήρας διαθέτει ένα σύνολο αριθμητικών τιμών, τα στατιστικά (*Stats*), που αντιπροσωπεύουν τις ικανότητες και τις αδυναμίες του, με σημαντικότερο από αυτά να είναι οι Πόντοι Ζωής (*Hit Points*). Για τους σκοπούς του πρωτοτύπου, ο Παίκτης μπορεί να επιλέξει τον αντίπαλο που θα αντιμετωπίζει σε κάθε μάχη από μια προκαθορισμένη λίστα τεσσάρων διαφορετικών εχθρών. Αυτοί οι εχθροί έχουν μοναδικά στατιστικά , προσδίδοντας έτσι μεγαλύτερο στρατηγικό βάθος στην εμπειρία.

Η ροή του παιχνιδιού είναι **σειριακή (*turn-based*)**, συνεπώς οι χαρακτήρες ενεργούν διαδοχικά βάση μιας προκαθορισμένης σειράς: Παίκτης -> Βοηθός -> Αντίπαλος. Η εκτέλεση των ενεργειών και των τριών χαρακτήρων συνιστά την ολοκλήρωση ενός **γύρου (*round*)** και την εκκίνηση ενός νέου.

Το παιχνίδι τερματίζεται όταν εκπληρωθεί μια από τις παρακάτω συνθήκες:

- **Ήττα Παίκτη**: Εκπληρώνεται όταν οι πόντοι ζωής του Παίκτη μηδενιστούν.
- **Νίκη Παίκτη**: Εκπληρώνεται όταν οι πόντοι ζωής του Αντιπάλου μηδενιστούν.

Αξίζει να σημειωθεί ότι ο μηδενισμός των πόντων ζωής του Βοηθού δεν οδηγεί στον τερματισμό του παιχνιδιού, αλλά έχει ως αποτέλεσμα απομάκρυνση του από τη μάχη.

Τέλος ορίζονται οι τύποι ενεργειών που μπορεί να εκτελέσει ένας χαρακτήρας κατά τη διάρκεια της σειράς του. Αυτές διακρίνονται σε τρεις κατηγορίες:

1. **Κινήσεις Ζημιάς (Επιθέσεις)**: Όταν ένας χαρακτήρας εκτελεί μια επίθεση, επιλέγει έναν στόχο από τον οποίο αφαιρείται ένας συγκεκριμένος αριθμός πόντων ζωής. Ο αριθμός αυτός καθορίζεται από τα στατιστικά των χαρακτήρων καθώς και των στατιστικών της κίνησης.
2. **Κινήσεις Αλλαγής Στατιστικών:** Με την εκτέλεση μιας τέτοιας κίνησης, ένας χαρακτήρας μεταβάλει συγκεκριμένα στατιστικά ενός επιλεγμένου χαρακτήρα για τρεις γύρους. Οι κινήσεις αυτές επιτρέπουν και την μεταβολή στατιστικών του ίδιου του χαρακτήρα.
3. **Κινήσεις Επαναφοράς Πόντων Ζωής:** Οι κινήσεις αυτής της κατηγορίας αποτελούν υπο-κατηγορία των κινήσεων αλλαγής στατιστικών, καθώς μεταβάλουν ένα συγκεκριμένο στατιστικό ενός χαρακτήρα. Οι κύριες διαφορές τους είναι ότι αλλάζουν πάντα το ίδιο στατιστικό (τους πόντους ζωής) και ότι η αλλαγή είναι μόνιμη.

3.3. Ορισμός των Δομικών Στοιχείων

Με την ολοκλήρωση του σταδίου της ιδέας, η διαδικασία μεταβαίνει στο στάδιο της επεξεργασίας. Σε αυτό το στάδιο οι κανόνες του παιχνιδιού πρέπει να μετατραπούν αρχικά σε δομικά στοιχεία και μηχανισμούς και στη συνέχεια αυτοί οι μηχανισμοί να υλοποιηθούν σε κώδικα. Στην παρούσα ενότητα, θα οριστούν τα βασικά δομικά στοιχεία που χρησιμοποιούν οι μηχανισμοί για να καθορίσουν την λειτουργία του παιχνιδιού.

3.3.1. Βασικά Δομικά Στοιχεία

Πριν τον ορισμό των κεντρικών οντοτήτων, είναι απαραίτητο να τεθούν δύο δομικά στοιχεία που χρησιμοποιούνται για την ομαλή αλληλεπίδραση τους.

Αρχικά, ορίζονται οι οντότητες **"Τύπος Ζημιάς" (*dmgTypes*)**. Αυτές οι οντότητες αναπαριστούν τον τρόπο με τον οποίο προκαλεί την ζημιά της μια επίθεση. Μπορούν να περιγραφούν με δύο συμβολικά χαρακτηριστικά:

- **Όνομα (*Name*)**: Το μοναδικό όνομα του τύπου.
- **Κατηγορία (*Category*)**: Η ευρύτερη κατηγορία στην οποία ανήκει. Ένας τύπος ζημιάς μπορεί να είναι είτε **φυσικός (*Physical*)** είτε **ειδικός (*Special*)**.

Στην *εικόνα 3.1* παρουσιάζονται οι 9 τύποι ζημιάς που συμπεριλαμβάνονται στο πρωτότυπο.

	Όνομα	Κατηγορία
1	Fire	Special
2	Water	Special
3	Grass	Special
4	Electric	Special
5	Ground	Special
6	Steel	Special

	Όνομα	Κατηγορία
7	Slashing	Physical
8	Piercing	Physical
9	Bludgeoning	Physical

Εικόνα 3.1: Οι τύποι ζημιάς (dmgType) του παιχνιδιού.

Το δεύτερο βασικό δομικό στοιχείο είναι οι οντότητες **"Μετρητής" (*Counter*)**. Είναι ένα σύνολο από πέντε σύνθετες οντότητες που παρακολουθούν την διάρκεια των προσωρινών αλλαγών που προκαλούν οι κινήσεις αλλαγής στατιστικών στα στατιστικά. Κάθε μετρητής έχει δύο χαρακτηριστικά:

- **Αριθμός (*Number*)**: Το μοναδικό αναγνωριστικό του αντίστοιχου μετρητή.
- **Μέτρηση (*Count*)**: Το χαρακτηριστικό που μετράει τους γύρους που έχουν ολοκληρωθεί από την εκτέλεση της κίνησης που παρακολουθεί.

3.3.2. Κεντρικά Δομικά Στοιχεία

Το σύστημα της μάχης περιστρέφεται γύρω από τους χαρακτήρες και τις κινήσεις που αυτοί εκτελούν.

Τα πρώτα κεντρικά δομικά στοιχεία του παιχνιδιού είναι οι οντότητες "**Χαρακτήρας" (*Character*)**. Το πρωτότυπο περιλαμβάνει τρεις τέτοιες οντότητες, τον **Παίκτη (*Player*)**, τον **Βοηθό (*Helper*)** και τον **Αντίπαλο (*Enemy*)**. Κάθε χαρακτήρας περιγράφεται από ένα σύνολο χαρακτηριστικών που καθορίζουν τις ικανότητες και αδυναμίες του:

- **<u>Αριθμητικά Χαρακτηριστικά:</u>**
  - **Πόντοι Ζωής (*HP*)**: Η μέγιστη αντοχή του χαρακτήρα απέναντι σε επιθέσεις.
  - **Τρέχοντες Πόντοι Ζωής (*CHP*)**: Οι πόντοι ζωής που διαθέτει ο χαρακτήρας σε κάθε δεδομένη στιγμή του παιχνιδιού.

- **Επίθεση (*ATK*)**: Η ικανότητα του χαρακτήρα να εκτελεί φυσικές επιθέσεις.
- **Άμυνα (*DEF*)**: Η ικανότητα του χαρακτήρα να αντιμετωπίζει φυσικές επιθέσεις.
- **Ειδική Επίθεση (*SP.ATK*)**: Η ικανότητα του χαρακτήρα να εκτελεί ειδικές επιθέσεις.
- **Ειδική Άμυνα (*SP.DEF*)**: Η ικανότητα του χαρακτήρα να αντιμετωπίζει ειδικές επιθέσεις.
- **Ταχύτητα (SPD)**: Η ταχύτητα εκτέλεσης επιθέσεων, καθώς και τα αντανακλαστικά για την αντιμετώπιση τους.

- **<u>Συμβολικά Χαρακτηριστικά:</u>**
  - **Όνομα (*Name*)**: Το αναγνωριστικό όνομα του χαρακτήρα.
  - **Αντιστάσεις (*RES*)**: Λίστα που περιέχει όλους τους *dmgTypes* τους οποίους ο χαρακτήρας είναι ικανός να αντιμετωπίζει αποτελεσματικά.
  - **Αδυναμίες (*VUL*)**: Λίστα που περιέχει όλους τους *dmgTypes* τους οποίους ο χαρακτήρας δεν είναι ικανός να αντιμετωπίζει αποτελεσματικά.

Οι ενέργειες που μπορούν να εκτελέσουν οι παραπάνω χαρακτήρες, αναπαρίστανται από τη δεύτερη κεντρική οντότητα του παιχνιδιού, την **"Κίνηση" (*Move*)**. Κάθε μεμονωμένη κίνηση είναι μια σύνθετη οντότητα, της οποίας τα χαρακτηριστικά ορίζουν τη φύση και τη λειτουργία της:

- **<u>Αριθμητικά Χαρακτηριστικά:</u>**
  - **Δύναμη (*movePower*)**: Η βασική ισχύς μιας επίθεσης ή το μέγεθος της αλλαγής ενός στατιστικού.
  - **Καθυστέρηση (*moveDelay*)**: Μια τιμή που τροποποιεί την ταχύτητα του επιτιθέμενου κατά την εκτέλεση μιας επίθεσης.
- **<u>Συμβολικά Χαρακτηριστικά:</u>**
  - **Όνομα (*moveName*)**: Το όνομα της κίνησης.
  - **Αναγνωριστικό (*moveNumber*)**: Ένας μοναδικός κωδικός για την αναγνώριση μιας κίνησης. Ο αριθμός αυτός ξεκινάει με 1 για τις επιθέσεις, με 2 για τις κινήσεις στατιστικών και με 3 για τις κινήσεις επαναφοράς πόντων ζωής.
  - **Μετρητής (*moveCounter*)**: Ο αναγνωριστικός αριθμός του μετρητή που χρησιμοποιείται από μια κίνηση στατιστικών.
  - **Στόχος (*moveTarget*)**: Ο χαρακτήρας-στόχος της κίνησης. Αυτό το χαρακτηριστικό αξιοποιείται αποκλειστικά από τις κινήσεις στατιστικών και πόντων ζωής όπου ο στόχος είναι πάντα ο ίδιος.
  - **Στατιστικά (*moveStats*)**: Καθορίζει ποια από τα στατιστικά του χαρακτήρα-στόχου θα μεταβάλει η κίνηση.
  - **Τύπος Ζημιάς (*moveDmgType*)**: Ένα σύνθετο χαρακτηριστικό το οποίο περιέχει μια ή περισσότερες οντότητες *dmgType* που ορίζουν τι τύπους ζημιάς προκαλεί η επίθεση.
  - **Διάλογος (moveDialogue)**: Το κείμενο που θα χρησιμοποιηθεί από τους μηχανισμούς για την ενημέρωση του παίκτη σχετικά με την κίνηση που εκτελέστηκε.

Στην *εικόνα 3.2* παρουσιάζεται ο κατάλογος με όλες τις διαθέσιμες κινήσεις του παιχνιδιού.

Κινήσεις Ζημιάς

moveNumber	moveName	movePower	moveDelay	moveType
101	Sword Slash	?	?	7
102	Hydro Pump	?	?	2,9
103	Arrows	?	?	6,8
104	Fire Pierce	?	?	1,8
105	Electric Hammer	?	?	4,9
106	Firebolt	?	?	1
107	Thunderbolt	?	?	4
108	Rock Smash	?	?	5,9
109	Thorns	?	?	3,8
110	Vines	?	?	3,8
111	Whip	?	?	7
112	Stone Paunch	?	?	5,9
113	Magnitude	?	?	5
114	Tackle	?	?	9
115	Watergun	?	?	2,9
116	Flames	?	?	1

Κινήσεις Αλλαγής Στατιστικών

moveNumber	moveName	movePower	moveStat	moveTarget	moveCounter
201	Shield	3	DEF	Player	0
202	Encouragement Spell	3	ATK	Player	1
203	Intimidation Spell	-3	ATK	Enemy	2
204	Distraction Spell	-3	DEF	Enemy	3
205	Evanescence	3	SPD	Enemy	4
206	Harden	2	DEF, SP.DEF	Enemy	4
207	Rage	2	ATK, SP.ATK	Enemy	4
208	Inner Power	2	SP.ATK, SPD	Enemy	4

Κινήσεις Επαναφοράς Πόντων Ζωής

moveNumber	moveName	movePower	moveStat	moveTarget
301	Heal Player	?	CHP	Player
302	Heal Helper	?	CHP	Helper
303	Heal Eenmy	?	CHP	Enemy

Εικόνα 3.2: Ο κατάλογος των διαθέσιμων κινήσεων του παιχνιδιού

Για τις κινήσεις ζημιάς είναι δυνατή η συμπλήρωση της τιμής του χαρακτηριστικού "*moveType*" καθώς το χαρακτηριστικό αυτό προσδιορίζει τον τύπο της ζημιάς που προκαλεί η επίθεση. Για παράδειγμα, η κίνηση *Electric Hammer*, έχει τους τύπους ζημιάς *Electric* (4) και *Bludgeoning* (9). Αυτό δικαιολογείται από το γεγονός ότι η πρόσκρουση του συγκεκριμένου όπλου σε έναν στόχο προκαλεί ζημιά τόσο μέσω της σύγκρουσης από τη φυσική κρούση (*Bludgeoning*) , όσο και μέσω του ηλεκτρισμού που παράγει (*Electric*). Οι αριθμητικές τιμές των χαρακτηριστικών "*movePower*" και "*moveDelay*" θα καθοριστούν κατά το στάδιο της βελτιστοποίησης.

Για τις κινήσεις μεταβολής στατιστικών, η τιμή του χαρακτηριστικού "*movePower*" καθορίζεται βάσει τον εξής κανόνα: "λαμβάνει την τιμή 3 στην περίπτωση μεταβολής ενός μόνο στατιστικού, και την τιμή 2 στην περίπτωση μεταβολής δύο στατιστικών".

Τέλος, για τις κινήσεις επαναφοράς πόντων ζωής δεν αποδίδεται τιμή στο χαρακτηριστικό "*movePower*", καθώς η τιμή του δίνεται από έναν αλγόριθμο παραγωγής τυχαίων αριθμών. Το εύρος τιμών του αλγορίθμου θα οριστικοποιηθεί κατά το στάδιο της βελτιστοποίησης.

3.3.3. Εσωτερική Οικονομία

Στο παρόν σύστημα, ο μοναδικός πόρος της εσωτερικής οικονομίας του παιχνιδιού είναι οι πόντοι ζωής, συγκεκριμένα το χαρακτηριστικό *CHP* κάθε χαρακτήρα καθορίζει την επιβίωση του στο παιχνίδι. Οι επιθέσεις λειτουργούν ως "στραγγιστές" , αφαιρώντας τον πόρο από το σύστημα, ενώ οι κινήσεις επαναφοράς πόντων ζωής λειτουργούν ως "πηγές", εισάγοντας τον ξανά στο παιχνίδι. Η διαχείριση αυτής της ροής αποτελεί τον πυρήνα της στρατηγικής του παίκτη.

3.3.4. Υποστηρικτικά Δομικά Στοιχεία για την Διαχείριση Ροής του Συστήματος Μάχης

Πέραν των οντοτήτων που αφορούν το κόσμο του παιχνιδιού, ο σχεδιασμός απαιτεί και ένα σύνολο οντοτήτων που διαχειρίζεται τη δομή της μάχης.

Αρχικά, η συμβολική οντότητα **"Επιλεγμένος Παίκτης" (*chosenEnemy*)** αποθηκεύει την επιλογή του χρήστη από τη λίστα των τεσσάρων διαθέσιμων εχθρών και λειτουργεί ως είσοδος για τον μηχανισμό που θα διαμορφώσει τα χαρακτηριστικά της οντότητας *Enemy*.

Για την ορθή διαχείριση της διαδοχής των σειρών ορίζεται η σύνθετη οντότητα **"Διαχειριστής Σειρών" (*turnManager*)**. Η οντότητα αυτή αποτελείται από δύο συμβολικά χαρακτηριστικά:

1. **Τρέχον Σειρά (*currentTurn*)**: Καταγράφει ποιος χαρακτήρας έχει σειρά να δράσει.
2. **Προηγούμενη Σειρά (*previousTurn*)**: Αποθηκεύει ποιος χαρακτήρας είχε προηγουμένως σειρά, επιτρέποντας στο σύστημα να επαναλάβει μια σειρά εάν κριθεί απαραίτητο.

Τα χαρακτηριστικά αυτά λαμβάνουν μια από τις προκαθορισμένες τιμές **"Σειρά Παίκτη" (*PT*)**, **"Σειρά Βοηθού" (*HT*)** και **"Σειρά Αντιπάλου" (*ET*)** σηματοδοτώντας ποιος χαρακτήρας έχει τον έλεγχο της τρέχουσας και της προηγούμενης σειράς.

Η συμβολική οντότητα **"Κατάσταση" (*State*)** ελέγχει την ροή των ενεργειών κάθε σειράς. Η οντότητα αυτή μπορεί να λάβει μια από τις παρακάτω συμβολικές τιμές που αναπαριστούν τις φάσεις στις οποίες μπορεί να βρίσκεται μια σειρά:

- **Προετοιμασία και Επιλογή Ενέργειας (Κατάσταση 'No State-NS):** Σε αυτή την φάση ο χαρακτήρας επιλέγει την κίνηση που σκοπεύει να εκτελέσει.
- **Εκτέλεση της Ενέργειάς (Κατάσταση 'Move Declaration-MD):** Σε αυτή τη φάση γίνονται οι κατάλληλοι υπολογισμοί για τα αποτελέσματα που θα έχει η κίνηση στο κόσμο του παιχνιδιού. Επιπλέον, ο παίκτης ενημερώνεται για την κίνηση που πρόκειται να εκτελεστεί, καθώς και για τους χαρακτήρες που συμπεριλαμβάνει.
- **Επίλυση και Ενημέρωση (Κατάσταση 'Aftermath-AM):** Ο κύριος στόχος της συγκεκριμένης φάσης είναι η διαχείριση των μακροπρόθεσμων συνεπειών της ενέργειας που εκτελέστηκε, καθώς και ο έλεγχος για ικανοποίηση κάποιας συνθήκης που θα θέσει το σύστημα σε μια ειδική κατάσταση.
- **Ειδική Κατάσταση (Κατάσταση 'Special State- ST**): Η συγκεκριμένη φάση ενεργοποιείται αποκλειστικά εφόσον εντοπιστεί η επαλήθευση μιας συνθήκης τερματισμού ή μια σημαντική αλλαγή στην κατάσταση της μάχης. Ο κύριος ρόλος της είναι η διαχείριση αυτών των ειδικών καταστάσεων και ο τερματισμός του παιχνιδιού εφόσον κριθεί αναγκαίο.

Η συμβολική οντότητα **"Μοτίβο Επίθεσης" (*atkPattern*)** χρησιμοποιείται για την αποθήκευση του επιτιθέμενου και του αμυνόμενου σε μια επίθεση. Οι τιμές που μπορεί να λάβει η συγκεκριμένη οντότητα είναι:

- **1**: Παίκτης -> Αντίπαλος
- **2:** Βοηθός -> Αντίπαλος
- **3:** Αντίπαλος -> Παίκτης
- **4:** Αντίπαλος -> Βοηθός

Η σημαία **"Μετρητής Ήταν Ενεργός" (*counterWasActive*)**, σηματοδοτεί αν η κίνηση στατιστικών που πρόκειται να εκτελεστεί έχει ήδη χρησιμοποιηθεί στους τελευταίους τρεις γύρους. Αν η σημαία είναι ενεργή, η σειρά του χαρακτήρα ακυρώνεται και πρέπει να επαναληφθεί ώστε να επιλεγεί μια διαφορετική κίνηση.

Επιπλέον των οντοτήτων που διαχειρίζονται τη ροή, το σύστημα χρησιμοποιεί και εξειδικευμένες υπολογιστικές οντότητες, οι οποίες είναι θεμελιώδεις για την εκτέλεση των επιθέσεων του παιχνιδιού.

Η συμβολική ενότητα **"Κατηγορία Ζημιάς" (*moveCategory*)** προκύπτει από τους *dmgTypes* της κίνησης που εκτελείται και καθορίζει την μαθηματική εξίσωση που θα

εφαρμοστεί για τον υπολογισμό της τιμής της αριθμητικής οντότητας **"Δύναμη Επίθεση" (*atkPow*)**.

Οι τροποποιητές **"Τροποποιητής Ταχύτητας" (*spdMod*)** και **"Τροποποιητής Αδυναμιών και Αντοχών" (*vulResMod*)** λειτουργούν ως ενδιάμεσες μεταβλητές που καθορίζουν την αποτελεσματικότητα μιας επίθεσης.

Το τελικό αποτέλεσμα των υπολογισμών που γίνονται για μια επίθεση αποθηκεύεται στην οντότητα "**Ζημιά Επίθεσης" (*moveDmg*)**. Η τιμή αυτής της οντότητας, που αντιστοιχεί στη συνολική ζημιά, αφαιρείται στη συνέχεια από τους πόντους ζωής (*CHP*) του αμυνόμενου.

Στην οντότητα **"Πόντοι Ζωής Κίνησης" (*moveHP*)** αποθηκεύεται ο αριθμός των πόντων ζωής που θα προστεθούν στους *CHP* του παίκτη-στόχου της αντίστοιχης κίνησης επαναφοράς πόντων ζωής.

Τέλος, η συμβολική οντότητα **"Ηττες" (*Losses*)** καταγράφει ποιοι χαρακτήρες έχουν ηττηθεί, πληροφορία κρίσιμη για τους μηχανισμούς που καθορίζουν το τελικό αποτέλεσμα του παιχνιδιού.

3.3.5. Δομικά Στοιχεία της Διεπαφής Χρήστη

Πέρα από τα δομικά στοιχεία που αξιοποιούνται από τους μηχανισμούς για τη εφαρμογή των κανόνων και τη λειτουργία του παιχνιδιού, απαραίτητα είναι και εκείνα που χρησιμοποιούνται για την ενημέρωση του παίκτη για την κατάσταση του κόσμου του παιχνιδιού.

Η οντότητα **"Κουτί Διαλόγου" (*dialogueBox*)** έχει τον βασικό ρόλο ενημέρωσης του παίκτη για τις ενέργειες που πραγματοποιούνται και τα αποτελέσματα τους στον κόσμο του παιχνιδιού.

Την ενημέρωση σχετικά με την κατάσταση στην οποία βρίσκεται ο παίκτης και ο βοηθός αναλαμβάνουν οι οντότητες **"Κουτί Παίκτη" (*playerBox*)** και **"Κουτί Βοηθού" (*helperBox*)**. Η καθεμία από αυτές διαθέτει τα χαρακτηριστικά "**Ονομα" (*Name*)** και **"Τρέχοντες Πόντοι Ζωής" (*CHP*)**, τα οποία αντικατοπτρίζουν δυναμικά τις αντίστοιχες τιμές των οντοτήτων του Παίκτη και του Βοηθού.

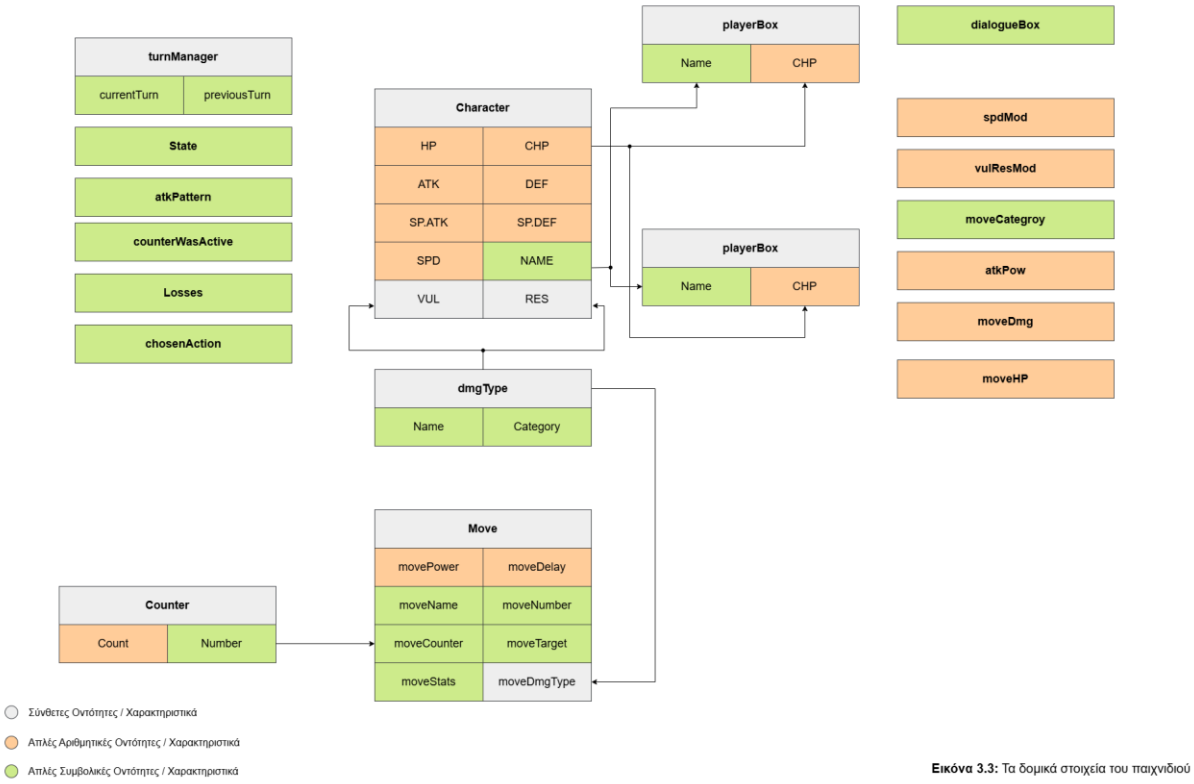
Στην *εικόνα 3.3* παρουσιάζονται όλα τα δομικά στοιχεία του παιχνιδιού καθώς και όποιες σχέσεις έχουν μεταξύ τους πριν την ενσωμάτωση των μηχανισμών.

3.4. Σχεδιασμός των Μηχανισμών

Έχοντας προσδιορίσει με πληρότητα το σύνολο των δομικών στοιχείων στη προηγούμενη ενότητα, η εργασία προχωράει στο δεύτερο και κρισιμότερο μέρος του σχεδιασμού, την ανάλυση των μηχανισμών. Το σύστημα της μάχης υλοποιείται μέσω ενός κύριου μηχανισμού που ενεργεί ως βρόγχος σε τέσσερις διακριτές φάσεις, εντός των οποίων αλληλεπιδρούν και οι δευτερεύοντες μηχανισμοί.

3.4.1. Φάση 0: Επιλογή Αντιπάλου και Αρχικοποίηση Μάχης

Πριν την ενεργοποίηση του κύριου μηχανισμού, προηγείται ο "***Μηχανισμός 1: Διαμόρφωση Μάχης***", ο οποίος υλοποιεί μια προκαταρκτική φάση του παιχνιδιού, κατά την οποία ο παίκτης καλείται να επιλέξει τον αντίπαλο που θα αντιμετωπίσει στην επερχόμενη μάχη.



Εικόνα 3.3: Τα δομικά στοιχεία του παιχνιδιού

<u>Μηχανισμός 1: Διαμόρφωση Μάχης</u>

Περιγραφή: Ο μηχανισμός αυτός διαχειρίζεται την επιλογή του αντιπάλου μέσω της διεπαφής χρήστη (*UI*) και χρησιμοποιεί την απόφαση του παίκτη για να αρχικοποιήσει τα στατιστικά της οντότητας *Enemy*. Στην συνέχεια ενεργοποιεί τον κεντρικό βρόγχο του παιχνιδιού.

Διεπαφή Χρήστη: Το σύστημα παρουσιάζει στον παίκτη μια οθόνη με ένα κατάλογο που περιέχει τις τέσσερις διαθέσιμες επιλογές αντιπάλων.

Συνθήκη: "Εάν το σύστημα ανιχνεύσει μια είσοδο από τον παίκτη που αντιστοιχεί στην τελική επιβεβαίωση μιας επιλογής από τον κατάλογο αντιπάλων".

- **Γεγονός:** Η εκπλήρωση της παραπάνω συνθήκη πυροδοτεί ένα γεγονός, το οποίο εκτελεί τις παρακάτω ενέργειες:
  - Η συμβολική οντότητα *chosenEnemy* λαμβάνει μια τιμή που αντιστοιχεί στο επιλεγμένο αντίπαλο.
  - Βάσει της τιμής της *chosenEnemy*, η οντότητα *Enemy* λαμβάνει τα αντίστοιχα στατιστικά από έναν προκαθορισμένο πίνακα δεδομένων.
  - Οι οντότητες *Player* και *Helper* αρχικοποιούνται από έναν προκαθορισμένο πίνακα δεδομένων.
  - Η διεπαφή χρήστη μεταβαίνει από την οθόνη επιλογής αντιπάλου στην οθόνη της μάχης και ο "**Μηχανισμός 2: Κεντρικός Βρόχος Ελέγχου Ροής**" ενεργοποιείται.

3.4.2. Φάση 1: Προετοιμασία και Επιλογή Ενέργειας (Κατάσταση 'No State' – NS)

Η εναρκτήρια φάση κάθε σειράς περιλαμβάνει δύο διακριτούς, διαδοχικούς μηχανισμούς που καθορίζουν τον ενεργό παίκτη και την επιλογή της κίνησης του.

Μηχανισμός 3: Καθορισμός Σειράς

Περιγραφή: Η κύρια λειτουργία του μηχανισμού είναι ο προσδιορισμός του ενεργού παίκτη, διασφαλίζοντας έτσι την προκαθορισμένη διαδοχή σειρών. Παράλληλα, ο μηχανισμός είναι υπεύθυνος για την εφαρμογή των απαραίτητων τροποποιήσεων όταν συνθήκες του παιχνιδιού επιβάλλουν την αλλαγή στην καθιερωμένη σειρά.

Συνθήκη 1: " Εάν (*turnManager.currentTurn* = "*PT*") ΚΑΙ (*losses* = "*No Losses*")"

- **Γεγονός:** Το *turnManager.currentTurn* λαμβάνει την τιμή "*HT*".

Συνθήκη 2: "Εάν ((*turnManager.currentTurn* = "*HT*") ΚΑΙ (*losses* = "*No Losses*")) Ή ((*turnManager.currentTurn* = "*PT*") ΚΑΙ (*losses* = "*Helper Lost*"))"

- **Γεγονός:** Το *turnManager.currentTurn* λαμβάνει την τιμή "*ET*".

Συνθήκη 3: "Εάν *turnManager.currentTurn* = "*ET*"".

- **Γεγονός:** Το *turnManager.currentTurn* λαμβάνει την τιμή "*PT*", σηματοδοτώντας την εκκίνηση ενός νέου γύρου.

Κοινό Γεγονός: Πριν από κάθε αλλαγή της οντότητας *turnManager.currentTurn* η εξερχόμενη τιμή της αποθηκεύεται στην *turnManager.previousTurn*.

Αποτέλεσμα στη Διεπαφή Χρήστη: Το αποτέλεσμα του μηχανισμού ενημερώνει άμεσα τον παίκτη μέσω της οντότητας *dialogueBox*, η οποία είτε ανακοινώνει την επόμενη σειρά είτε παρουσιάζει τον κατάλογο των διαθέσιμων κινήσεων του παίκτη αν είναι η σειρά του.

Μηχανισμός 4: Επιλογή Κίνησης

Περιγραφή: Αφού καθοριστεί ο ενεργός χαρακτήρας για την παρούσα σειρά, ο μηχανισμός αυτός καταγράφει την κίνηση που επέλεξε να εκτελέσει.

Συνθήκη: Η κύρια συνθήκη για την ενεργοποίηση αυτού του μηχανισμού είναι η επιβεβαίωση του παίκτη (το πάτημα του πλήκτρου "*enter*"), ανεξάρτητα από το ποιος χαρακτήρας έχει σειρά.

- **Γεγονός:** Όταν η συνθήκη πληροίτε, η οντότητα *chosenEnemy* λαμβάνει την τιμή του *Move.moveNumber* της κίνησης που επιλέχθηκε. Η προέλευση αυτής της τιμής διαφοροποιείται ως εξής:
  - **Για τον παίκτη:** Η τιμή προέρχεται απευθείας από την επιλογή του χρήστη στη διεπαφή.
  - **Για τον βοηθό:** Η τιμή είναι το αποτέλεσμα του συστήματος τεχνητής νοημοσύνης.
  - **Για τον αντίπαλο:** Η τιμή καθορίζεται από έναν αλγόριθμο παραγωγής τυχαίων αριθμών.

Κοινό Γεγονός: Αφότου καταγραφεί το αναγνωριστικό της επιλεγμένης κίνησης στην οντότητα *chosenEnemy*, η τιμή της οντότητας *State* αλλάζει σε Move Declaration (*MD*), σηματοδοτώντας το τέλος της φάσης της προετοιμασίας και την έναρξη της εκτέλεσης της ενέργειας.

3.4.3. Φάση 2: Εκτέλεση και Υπολογισμός Ενέργειας (Κατάσταση 'Move Declaration' – MD)

Κατά τη διάρκεια αυτής της φάσης, ενεργοποιούνται οι αντίστοιχοι μηχανισμοί για την εκτέλεση της επιλεγμένης κίνησης και τον υπολογισμό των αποτελεσμάτων της. Η μετάβαση σε αυτή τη φάση προϋποθέτει την ανάθεση της τιμής "*MD*" στην οντότητα *State* και την επιβεβαίωση από τον παίκτη.

Μηχανισμός 5: Καθορισμός Πλαισίου Επίθεσης

Περιγραφή: Ο συγκεκριμένος μηχανισμός λειτουργεί αποκλειστικά για τις επιθετικές κινήσεις. Ο ρόλος του είναι ο προσδιορισμός του επιτιθέμενου και του αμυνόμενου, προτού ενεργοποιηθούν οι μηχανισμοί υπολογισμού της ζημιάς.

Συνθήκη 1: "Εάν *turnManager.currentTurn* = "*PT*""
- **Γεγονός:** Η οντότητα *atkPattern* λαμβάνει την τιμή 1, ορίζοντας το μοτίβο επίθεσης "Παίκτης -> Αντίπαλος".

Συνθήκη 2: "Εάν *turnManager.currentTurn* = "*HT*""
- **Γεγονός:** Η οντότητα *atkPattern* λαμβάνει την τιμή 2, ορίζοντας το μοτίβο επίθεσης "Βοηθός -> Αντίπαλος".

Συνθήκη 3: "Εάν *turnManager.currentTurn* = "*ET*""
- **Γεγονός:** Ένας αλγόριθμος παραγωγής τυχαίων αριθμών δίνει μια ακέραια τιμή (1 ή 2). Η οντότητα *atkPattern* λαμβάνει την τιμή 2 + [τυχαίος αριθμός], στοχεύοντας είτε τον Παίκτη (τιμή 3) είτε τον Βοηθό (τιμή 4).

Μηχανισμός 6: Εκτέλεση Κίνησης

Περιγραφή: Ο μηχανισμός αυτός λαμβάνοντας το αναγνωριστικό της επιλεγμένης κίνησης, αναγνωρίζει τον τύπο της και ενεργοποιεί την κατάλληλη αλυσίδα υπο-μηχανισμών για τον υπολογισμό του αποτελέσματος της.

Συνθήκη 1: "Εάν η τιμή του *chosenAction* ξεκινάει με '1'"
- **Γεγονός:** Ενεργοποιούνται διαδοχικά οι μηχανισμοί 6.1.α – 6.1.ε

Συνθήκη 2: "Εάν η τιμή του *chosenAction* ξεκινάει με '2'"
- **Γεγονός:** Ενεργοποιούνται διαδοχικά οι μηχανισμοί 6.2.α.-6.2.β

Συνθήκη 3: "Εάν η τιμή του *chosenAction* ξεκινάει με '3'"
- **Γεγονός:** Ενεργοποιείται ο μηχανισμός 6.3.

Μηχανισμός 6.1.α: Υπολογισμός Τροποποιητή Ταχύτητας (*spdMod*)

Περιγραφή: Ο μηχανισμός αυτός συγκρίνει την ταχύτητα του επιτιθέμενου με αυτή του αμυνόμενου για να προσδιορίσει αν η επίθεση του θα έχει πλεονέκτημα ή μειονέκτημα.

Συνθήκη 1: "Εάν (*SPD_Επιτιθέμενου – Move.moveDelay*) > *SPD_Αμυνόμενου*"
- **Γεγονός:** Η οντότητα *spdMod* λαμβάνει την τιμή 1.

Συνθήκη 2: "Εάν (*SPD_Επιτιθέμενου – Move.moveDelay*) = *SPD_Αμυνόμενου* "
- **Γεγονός:** Η οντότητα *spdMod* λαμβάνει την τιμή 0.

Συνθήκη 3: "Εάν (*SPD_Επιτιθέμενου – Move.moveDelay*) < *SPD_Αμυνόμενου* "
- **Γεγονός:** Η οντότητα *spdMod* λαμβάνει την τιμή -1.

Μηχανισμός 6.1.β: Υπολογισμός Τροποποιητή Αδυναμιών/Αντοχών (*vulResMod*)

Περιγραφή: Ο συγκεκριμένος μηχανισμός ποσοτικοποιεί την αποτελεσματικότητα των τύπων ζημιάς της επιλεγμένης κίνησης έναντι του στόχου.

Συνθήκη 1: "Εάν ο αριθμός των κοινών τύπων του *Move.moveDmgType* με τη λίστα *VUL* είναι κατά δύο μεγαλύτερος από τον αριθμό των κοινών τύπων με την λίστα *RES*"
- **Γεγονός:** Η οντότητα *vulResMod* λαμβάνει την τιμή 4.

Συνθήκη 2: "Εάν ο αριθμός των κοινών τύπων του *Move.moveDmgType* με τη λίστα *VUL* είναι κατά ένα μεγαλύτερος από τον αριθμό των κοινών τύπων με την λίστα *RES*"
- **Γεγονός:** Η οντότητα *vulResMod* λαμβάνει την τιμή 2.

Συνθήκη 3: "Εάν ο αριθμός των κοινών τύπων του *Move.moveDmgType* με τη λίστα *VUL* είναι ίσος με τον αριθμό των κοινών τύπων με τη λίστα *RES*".
- **Γεγονός:** Η οντότητα *vulResMod* λαμβάνει την τιμή 1.

Συνθήκη 4: "Εάν ο αριθμός των κοινών τύπων του *Move.moveDmgType* με τη λίστα *RES* είναι κατά ένα μεγαλύτερος από τον αριθμό των κοινών τύπων με τη λίστα *VUL*"
- **Γεγονός:** Η οντότητα *vulResMod* λαμβάνει την τιμή 0.5.

Συνθήκη 5: "Εάν ο αριθμός των κοινών τύπων του *Move.moveDmgType* με τη λίστα *RES* είναι κατά δύο μεγαλύτερος από τον αριθμό των κοινών τύπων με τη λίστα *VUL*"

- **Γεγονός:** Η οντότητα *vulResMod* λαμβάνει την τιμή 0.25.

Μηχανισμός 6.1.γ: Καθορισμός Κατηγορίας Επίθεσης (*moveCategory*)

Περιγραφή: Ο μηχανισμός αυτός εξετάζει το χαρακτηριστικό *dmgType.Category* όλων των τύπων ζημιάς της επιλεγμένης επίθεσης για να ορίσει την κατηγορία της.

Συνθήκη 1: "Εάν ισχύει *dmgType.Category* = "*Physical*" για όλες τις τιμές του χαρακτηριστικού *Move.moveDmgType* της επίθεσης"

- **Γεγονός:** Το *moveCategory* λαμβάνει την τιμή "*Physical*"

Συνθήκη 2: "Εάν ισχύει *dmgType.Category* = "*Special*" για όλες τις τιμές του χαρακτηριστικού *Move.moveDmgType* της επίθεσης"

- **Γεγονός:** Το *moveCategory* λαμβάνει την τιμή "*Special*"

Συνθήκη 3: "Εάν οι τύποι του *Move.moveDmgType* έχουν "*Special*" και "*Physical*" τιμές στο χαρακτηριστικό *dmgType.Category* τους"

- **Γεγονός:** Το *moveCategory* λαμβάνει την τιμή "*Both*".

Μηχανισμός 6.1.δ: Υπολογισμός Βασικής Δύναμης (*atkPow*)

Περιγραφή: Ο μηχανισμός αυτός υλοποιεί μια αριθμητική σχέση, που διαφοροποιείται ανάλογα με την κατηγορία της επίθεσης. Αυτή η αριθμητική σχέση συνδυάζει τα στατιστικά των χαρακτήρων και το *Move.movePower* για να υπολογίσει την βασική δύναμη της επίθεσης.

Συνθήκη 1: "Εάν *moveCategory* = "*Physical*""

- **Γεγονός:** *atkPow* = *ATK_Επιτιθέμενου* – 0.5**DEF_Αμυνόμενου* + *Move.movePower*. (3.1)

Συνθήκη 2: "Εάν *moveCategory* = "*Special*""

- **Γεγονός:** *atkPow* = *SP.ATK_Επιτιθέμενου* – 0.5**SP.DEF_Αμυνόμενου* + *Move.movePower*. (3.2)

Συνθήκη 3: "Εάν *moveCategory* = "*Both*""

- **Γεγονός:** *atkPow* = 0.5*(*ATK_Επιτιθέμενου* – 0.5* *DEF_Αμυνόμενου*) + 0.5*(*SP.ATK_Επιτιθέμενου* – *SP.DEF_Αμυνόμενου*) + *Move.movePower*. (3.3)

Κοινό Γεγονός: Το τελικό αποτέλεσμα στρογγυλοποιείται προς τα πάνω.

6.1.ε: Υπολογισμός και Εφαρμογή της Τελικής Ζημιάς (*moveDmg*)

Περιγραφή: Ο τελευταίος μηχανισμός της αλυσίδας που ορίζει τη συνολική τιμή της ζημιάς της επίθεσης.

Κοινό Γεγονός: Η τελική εξίσωση που συνδυάζει όλα τα προηγούμενα αποτελέσματα είναι: *moveDmg* = (*atkPow***vulResMod* + 0.25*(*atkPow***spdMod*))*0.5. (3.4)

Το τελικό αποτέλεσμα στρογγυλοποιείται προς τα πάνω.

Συνθήκη 1: "Εάν το *chosenAction* είναι 111, 113, ή 114"

- **Γεγονός:** Ένας αλγόριθμος παραγωγής τυχαίων αριθμών παράγει ένα αριθμό (1 ή 2). Αν ο παραγόμενος αριθμός είναι 1 τότε το *moveDmg* διπλασιάζεται, αλλιώς υποδιπλασιάζεται.

Συνθήκη 2: "Εάν το *chosenAction* είναι 103"

- **Γεγονός:** Ένας αλγόριθμος παραγωγής τυχαίων αριθμών παράγει έναν αριθμό (1 ή 2 ή 3). Το *moveDmg* πολλαπλασιάζεται με αυτό τον αριθμό.

Τελικό Γεγονός: Σε όλες τις περιπτώσεις, η τελική τιμή του *moveDmg* αφαιρείται από το *CHP_Αμυνόμενου*, λειτουργώντας ως "στραγγιστής" για τον πόρο των πόντων ζωής.

Μηχανισμός 6.2.α: Έλεγχος Μετρητών (*counterWasActive*)

Περιγραφή: Αυτός ο μηχανισμός ελέγχει εάν μια κίνηση στατιστικών μπορεί να εκτελεστεί, εξετάζοντας αν τα στατιστικά που επηρεάζει είναι ήδη τροποποιημένα.

Συνθήκη 1: "Εάν (*Move.moveCounter* = *Counter.Number*) ΚΑΙ (*Counter.Count* >0) "

- **Γεγονός:** Η σημαία *counterWasActive* λαμβάνει την τιμή *true*. Η διαδικασία τερματίζεται εδώ, καθώς τα στατιστικά είναι ήδη τροποποιημένα.

Συνθήκη 2: "Εάν (*Move.moveCounter* = *Counter.Number*) ΚΑΙ (*Counter.Count* = 0)"

- **Γεγονός:** Η σημαία *counterWasActive* παραμένει *false*, επιτρέποντας στη διαδικασία να συνεχίσει.

Μηχανισμός 6.2.β: Αλλαγή Στατιστικών

Περιγραφή: Εάν ο προηγούμενος έλεγχος είναι επιτυχής, ο μηχανισμός αυτός εφαρμόζει την αλλαγή στα στατιστικά του στόχου και ενεργοποιεί τον αντίστοιχο μετρητή που την παρακολουθεί.

Συνθήκη: "Εάν *counterWasActive* = "*false*""

- **Γεγονός:** Τα χαρακτηριστικά του στόχου (*Move.moveTarget*), που προσδιορίζονται από το *Move.moveStat*, τροποποιούνται κατά τη τιμή του *Move.movePower*. Αμέσως μετά, η τιμή του *Counter.Count* του αντίστοιχου μετρητή αυξάνεται κατά 1, ξεκινώντας την τριών γύρων περίοδο επαναφοράς.

Μηχανισμός 6.3: Εκτέλεση Κίνησης Επαναφοράς Πόντων Ζωής (*moveHP*)

Περιγραφή: Ο μηχανισμός προσθέτει πόντους ζωής στο στόχο, διασφαλίζοντας ότι δεν θα υπάρξει υπερχείλιση.

Γεγονός: Ένας αλγόριθμος παραγωγής τυχαίων αριθμών ορίζει την αρχική τιμή της οντότητας *moveHP*.

Γεγονός: Η τιμή της *moveHP* προστίθεται στο *CHP* του χαρακτήρα στόχου (*moveTarget*).

Συνθήκη: "Εάν *CHP_στόχου* > *HP_στόχου*"

- **Γεγονός:** Η τιμή του *moveHp* υπολογίζεται ξανά ώστε να αντικατοπτρίζει μόνο τους πόντους ζωής που πραγματικά αναπληρώθηκαν. Ο υπολογισμός αυτός γίνεται μέσω της εξίσωσης: *moveHP* = *HP_στόχου* – *CHP_στόχου* + *moveHP* (3.5). Ακολούθως, το *CHP_στόχου* γίνεται ίσο με το *HP_στόχου*.

3.4.4. Φάση 3: Επίλυση και Ενημέρωση (Κατάσταση 'Aftermath' – AM)

Η τρίτη φάση του σχεδιασμού αποτελείται από έναν μηχανισμό που εκτελεί δύο λειτουργίες. Αρχικά, διαχειρίζεται τις μακροπρόθεσμες συνέπειες της ενέργειας που μόλις εκτελέστηκε και στη συνέχεια αξιολογεί αν η κατάσταση του παιχνιδιού έχει μεταβληθεί αρκετά ώστε να απαιτείται η μετάβαση σε μια ειδική κατάσταση. Η φάση ενεργοποιείται όταν υπάρχει επιβεβαίωση συνέχισης από τον παίκτη και η οντότητα *State* έχει λάβει την τιμή "*AM*".

Μηχανισμός 7: Διαχείριση της Κατάστασης για την Ολοκλήρωση της Σειράς

Περιγραφή: Ο παρόν μηχανισμός αποτελείται από τρεις υπο-μηχανισμούς. Ο πρώτος από αυτούς ενημερώνει τους ενεργούς μετρητές (*μηχανισμός 7.1*), ο δεύτερος επαναφέρει τα αλλαγμένα στατιστικά στις αρχικές τους τιμές (*μηχανισμός 7.2*) και ο τελευταίος ελέγχει τις συνθήκες που οδηγούν το σύστημα σε μια ειδική κατάσταση (*μηχανισμό 7.3*).

Μηχανισμός 7.1: Ενημέρωση Ενεργών Μετρητών

Περιγραφή: Αυτός ο υπο-μηχανισμός ελέγχει τη τιμή του χαρακτηριστικού *Counter.Count* όλων των μετρητών και την αυξάνει αν είναι μεγαλύτερη του ένα και μικρότερη του τέσσερα.

Συνθήκη: "Εάν (*Counter.Count* > 0) ΚΑΙ *(Counter.Count* <4)"

- **Γεγονός:** Η τιμή του χαρακτηριστικού *Counter.Count* του αντίστοιχου μετρητή αυξάνεται κατά ένα.

Μηχανισμός 7.2: Επαναφορά των Αλλαγμένων Στατιστικών

Περιγραφή: Αυτός ο υπο-μηχανισμός ελέγχει τους μετρητές των κινήσεων αλλαγής στατιστικών και αν εντοπίσει ότι μια κίνηση εκτελέστηκε πριν από τρεις γύρους, επαναφέρει τα στατιστικά που επεξεργάστηκε πίσω στις αρχικές τιμές τους.

Συνθήκη 1: "Αν (*turnManager.currentTurn* = "*PL*") ΚΑΙ (*Counter.Number* = 0) ΚΑΙ (*Counter.Count* = 4)"

- **Γεγονός:** Η τιμή του χαρακτηριστικού *Player.DEF* μειώνεται κατά 3.

Συνθήκη 2: "Αν (*turnManager.currentTurn* = "*HT*") ΚΑΙ (*Counter.Number* = 1) ΚΑΙ (*Counter.Count* = 4)"

- **Γεγονός:** Η τιμή του χαρακτηριστικού *Player.ATK* μειώνεται κατά 3.

Συνθήκη 3: "Αν (*turnManager.currentTurn* = "*HT*") ΚΑΙ (*Counter.Number* = 2) ΚΑΙ (*Counter.Count* = 4)"

- **Γεγονός:** Η τιμή του χαρακτηριστικού *Enemy.ATK* αυξάνεται κατά 3.

Συνθήκη 4: "Αν (*turnManager.currentTurn* = "*HT*") ΚΑΙ (*Counter.Number* = 3) ΚΑΙ (*Counter.Count* = 4)"

- **Γεγονός:** Η τιμή του χαρακτηριστικού *Enemy.DEF* αυξάνεται κατά 3.

Συνθήκη 5: "Αν (*turnManager.currentTurn* = "*ET*") ΚΑΙ (*Counter.Number* = 4) ΚΑΙ (*Counter.Count* = 4) ΚΑΙ (*chosenEnemy* =1)"

- **Γεγονός:** Η τιμή του χαρακτηριστικού *Enemy.SPD* μειώνεται κατά 3.

Συνθήκη 6: "Αν (*turnManager.currentTurn* = "*ET*") ΚΑΙ (*Counter.Number* = 4) ΚΑΙ (*Counter.Count* = 4) ΚΑΙ (*chosenEnemy* =2)"

- **Γεγονός:** Η τιμή των χαρακτηριστικών *Enemy.DEF* και *Enemy.SP.DEF* μειώνεται κατά 2"

Συνθήκη 7: "Αν (*turnManager.currentTurn* = "*ET*") ΚΑΙ (*Counter.Number* = 4) ΚΑΙ (*Counter.Count* = 4) ΚΑΙ (*chosenEnemy* =3)"

- **Γεγονός:** Η τιμή των χαρακτηριστικών *Enemy.ATK* και *Enemy.SP_ATK* μειώνεται κατά 2"

Συνθήκη 8: "Αν (*turnManager.currentTurn* = "*ET*") ΚΑΙ (*Counter.Number* = 4) ΚΑΙ (*Counter.Count* = 4) ΚΑΙ (*chosenEnemy* =4)"

- **Γεγονός:** Η τιμή των χαρακτηριστικών *Enemy.SP_ATK* και *Enemy.SPD* μειώνεται κατά 2"

Μηχανισμός 7.3: Έλεγχος Επόμενης Κατάστασης

Περιγραφή: Ο υπο-μηχανισμός αυτός ελέγχει αν το σύστημα θα μεταβεί στην επόμενη σειρά ή αν θα πρέπει να εισέρθει σε μια ειδική κατάσταση.

Συνθήκη 1: "Εάν (*CHP_Όλων* > 0) Ή (*Player.CHP* > 0 ΚΑΙ *Enemy.CHP*>0 ΚΑΙ *Losses* = "*HL*")"

- **Γεγονός:** Η οντότητα *State* λαμβάνει την τιμή "*NS*" και ο κεντρικός βρόχος προετοιμάζεται για την επόμενη σειρά.

Συνθήκη 2: "Εάν *Player.CHP* < 1"

- **Γεγονός:** Οι οντότητες *State* και *Losses* λαμβάνουν τις τιμές "*ST*" και "*PL*" αντίστοιχα.

Συνθήκη 3: "Εάν (*Helper.CHP* < 1) ΚΑΙ (*Losses* = "*NL*")"

- **Γεγονός:** Οι οντότητες *State* και *Losses* λαμβάνουν τις τιμές "*ST*" και "*HL*" αντίστοιχα.

Συνθήκη 4: "Εάν *Enemy.CHP* < 1"

- **Γεγονός:** Οι οντότητες *State* και *Losses* λαμβάνουν τις τιμές "*ST*" και "*EL*" αντίστοιχα.

3.4.5. Φάση 4: Διαχείριση Ειδικών Καταστάσεων (Κατάσταση 'Special State' – ST)

Αυτή είναι η τελική φάση του βρόχου, η οποία ενεργοποιείται από μια είσοδο επιβεβαίωσης του παίκτη και μόνο αν ανιχνευτεί μια συνθήκη τερματισμού ή μια σημαντική αλλαγή στην κατάσταση της μάχης.

Μηχανισμός 8: Διαχείριση Αποτελέσματος

Περιγραφή: Ο μηχανισμός αυτός ερμηνεύει την τιμή της οντότητας *losses* και εκτελεί ένα γεγονός που είτε τερματίζει τη μάχη είτε την συνεχίζει υπο νέες συνθήκες.

Συνθήκη 1: "Εάν *losses* = "*PL*""

- **Γεγονός:** Η οντότητα *dialogueBox* λαμβάνει ένα μήνυμα που ενημερώνει τον παίκτη για την ήττα του. Η κεντρική διαδικασία του παιχνιδιού τερματίζεται.

Συνθήκη 2: "Εάν *losses* = "*EL*"".

- **Γεγονός:** Η οντότητα *dialogueBox* λαμβάνει ένα μήνυμα που ενημερώνει τον παίκτη για τη νίκη του. Το παιχνίδι μεταβαίνει στην οθόνη επιλογής αντιπάλου και την "*Φάση 0: Επιλογή Αντιπάλου και Αρχικοποίηση Μάχης*" για την επόμενη μάχη.

Συνθήκη 3: "Εάν *losses* = "*HL*"".

- **Γεγονός:** Η οντότητα *dialogueBox* λαμβάνει ένα μήνυμα που ενημερώνει τον παίκτη ότι ο Βοηθός έχει ηττηθεί και αποχωρεί από τη μάχη. Η οντότητα *State* παίρνει την τιμή "NS", επιτρέποντας στο παιχνίδι να συνεχιστεί με τους εναπομείναντες χαρακτήρες.

3.4.6. Μηχανισμός 9: Ενημέρωση του Κόσμου του Παιχνιδιού και της Διεπαφής

Στην σχεδίαση υπάρχει ένας μηχανισμός που δεν λειτουργεί μόνο σε μια φάση, αλλά στο τέλος κάθε φάσης. Η λειτουργία του είναι η ενημέρωση του παίκτη σχετικά με την κίνηση που εκτελείται στην τρέχον σειρά, καθώς και των αποτελεσμάτων της.

Συνθήκη 1: "Εάν η κίνηση που εκτελέστηκε ήταν τύπου "Επίθεσης" Ή "Επαναφοράς Πόντων Ζωής"".

- **Γεγονός:** Ενημερώνονται οι τιμές των *playerBox.CHP* και *helperBox.CHP* για να αντικατοπτρίζουν τις νέες τιμές των αντίστοιχων χαρακτηριστικών.

Συνθήκη 2: "Εάν *State* = "*MD*""

- **Γεγονός:** Η τιμή του *dialogueBox* μετατρέπεται σε ένα κείμενο που συμπεριλαμβάνει τα χαρακτηριστικά *Name_Επιτιθέμενου*, *Name_Αμυνόμενου, Move.moveDialogue* και *Move.moveName* ώστε να περιγραφεί στον παίκτη η ενέργεια που μόλις εκτελέστηκε. Επιπλέον η τιμή της οντότητας *State* αλλάζει σε "*AM*".

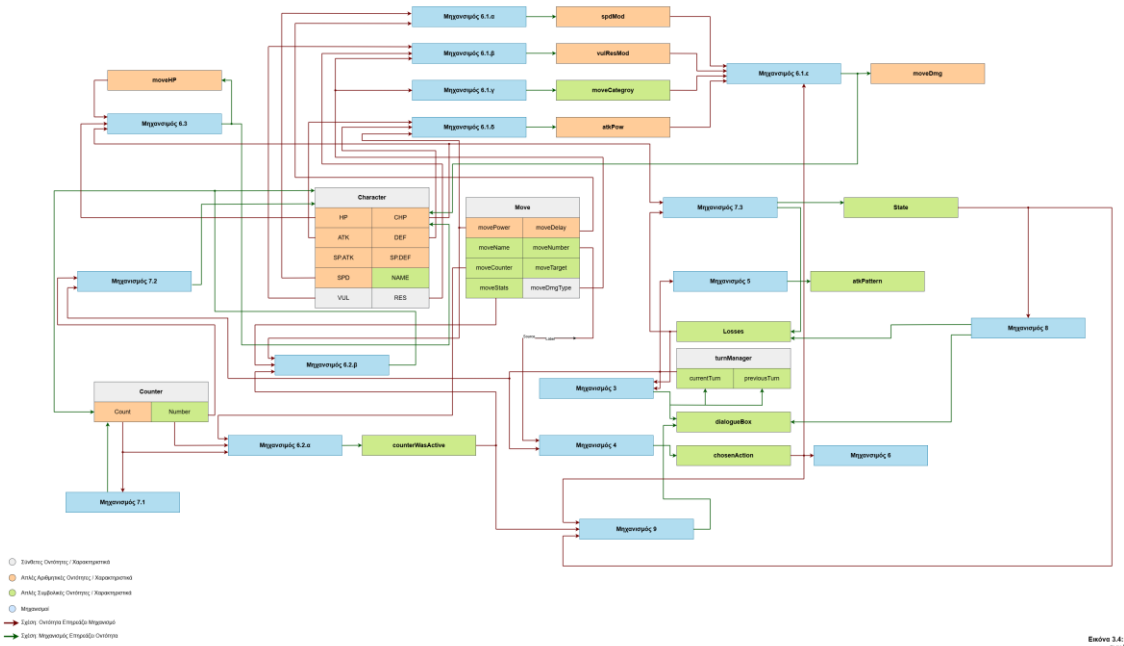
Συνθήκη 3: "Εάν (*State* = "*AM*") ΚΑΙ (*counterWasActive* = "*true*")".

- **Γεγονός:** Συντίθεται ένα μήνυμα που ενημερώνει τον παίκτη για την αποτυχία εκτέλεσης της κίνησης και η οντότητα *State* λαμβάνει την τιμή "*NS*" για να επαναληφθεί η σειρά.

Συνθήκη 4: "Εάν (*State* = "*AM*") ΚΑΙ (*counterWasActive* = "false")".

- **Γεγονός:** Ανάλογα με τον τύπο της κίνησης, συντίθεται το κατάλληλο μήνυμα που περιγράφει το αποτέλεσμα: απώλεια HP (κάνοντας χρήση της τιμή του *moveDmg*), την αλλαγή στατιστικών (χρησιμοποιώντας τις τιμές των *Move.moveStat* και *move.movePower*) ή την αναπλήρωση HP (κάνοντας χρήση τη τιμή του *moveHP*).

Στην *εικόνα 3.4.* παρουσιάζονται οι σχέσεις μεταξύ των μηχανισμών και των οντοτήτων του συστήματος.



Εικόνα 3.4: Οι σχέσεις μεταξύ των Οντοτήτων και των Μηχανισμών του Συστήματος

3.5. Μετατροπή των Μηχανισμών σε Κώδικα

Η παρούσα ενότητα εστιάζει στην κωδικοποίηση των μηχανισμών που αναλύθηκαν προηγουμένως. Η υλοποίηση πραγματοποιήθηκε στο περιβάλλον της μηχανής "*Godot*", με χρήση της γλώσσας προγραμματισμού *C#*. Ο κώδικας που παρατίθεται στις επόμενες υποενότητες αποτελεί μια απλοποιημένη εκδοχή του πραγματικού κώδικα, αποσκοπώντας στη καλύτερη αναγνωσιμότητα και κατανόηση της υλοποίησης.

3.5.1. Κεντρικά Δομικά Στοιχεία

Αρχικά, πρέπει να υλοποιηθούν οι οντότητες *Character* και *Move*, καθώς αποτελούν τα θεμελιώδη δομικά στοιχεία πάνω στα οποία δομείται το σύστημα μάχης του πρωτοτύπου.

Για την αναπαράσταση της οντότητας *Character*, υλοποιείται μια κλάση (*class*) μέσω της οποίας στην συνέχεια θα είναι δυνατή η δημιουργία αντικειμένων για τους χαρακτήρες του παιχνιδιού.

```
//Η κλάση κληρονομεί από την κλάση Node της Godot, ώστε να μπορεί να
//προστεθεί στο Scene Tree, να έχει θυγατρικούς κόμβους και να αλληλεπιδρά
//με το υπόλοιπο παιχνίδι

public partial class character : Node


//Ορισμός των ιδιωτικών πεδίων της κλάσης.
      private string _name;
      private int _hp;
      //ορισμός και των υπόλοιπων χαρακτηριστικών
```

```
//Ορισμός των δημόσιων ιδιοτήτων της κλάσης. Κάθε ιδιότητα χρησιμοποιεί
έναν getter για να επιστρέφει την τιμή του αντίστοιχου ιδιωτικού πεδίου και
έναν setter για να την ενημερώνει.
      public string Name {
            get {return _name;} //Ο getter
            set {_name = value;}}

//Ένας κατασκευαστής(constructor) που καλείται όταν δημιουργείται ένα νέο
//αντικείμενο της κλάσης και αρχικοποιεί το αντικείμενο.
      public character (string name, int hp, … ){
            Name = name;
            HP = hp;
            …
      }
}
```

Κώδικας 3.1: Κλάση και Κατασκευαστής για την οντότητας Character.

Για την αποθήκευση και διαχείριση των δεδομένων που αφορούν την οντότητα *Move*, υιοθετείται η δομή **JSON (*JavaScript Object Notation*)**. Οι κινήσεις του παιχνιδιού αναπαρίστανται από δύο **πίνακες JSON (*JSON Arrays*)**, ένας για τις επιθέσεις και ο δεύτερος για τις κινήσεις αλλαγής στατιστικών και επαναφοράς πόντων ζωής. Κάθε πίνακας περιέχει, ως στοιχεία διαχωρισμένα με κόμματα, **αντικείμενα JSON (*JSON Objects*)** που αναπαριστούν τα χαρακτηριστικά μιας κίνησης. Τα μοναδικά χαρακτηριστικά κάθε κίνησης περιγράφονται από ένα **ζεύγος ονόματος/τιμής (*name/value pair*)** [19].

```
[
      {
            “moveNumber”: 101,
            “Name”: “Sword Slash”,
            “movePower”: [TBA],
            “moveDelay”: [TBA],
            “moveType:” 7,
            “dialogue1”: “SLASHES”,
            “dialogue2”: “WITH THEIR SWORD!”
      },
      …
]
```

Κώδικας 4.2(α): Παράδειγμα της δομής του JSON αρχείου για τις κινήσεις ζημιάς.

```
[
      {
            “moveNumber”: 201,
            “Name”: “Shield”,
            “moveStat”: “DEF”,
            “moveTarget”: “Player”,
            “movePower”: 3,
            “moveCounter”: 0,
            “dialogue1”: “RAISES THEIR SHIELD”,
            “dialogue2”: “’S DEF RAISES BY 3”,
            “dialogue3”: “YOUR SHIELD IS ALREADY UP!”
      },
      …
]
```

Κώδικας 4.2(β): Παράδειγμα της δομής του JSON αρχείου για τις κινήσεις αλλαγής στατιστικών και επαναφοράς πόντων ζωής.

3.5.2. Οθόνες Επιλογής Αντιπάλου και Μάχης

Οι οθόνες επιλογής αντιπάλου και μάχης αποτελούν τα κεντρικά περιβάλλοντα αλληλεπίδρασης του παίκτη με το παιχνίδι.

Εικόνα 3.5: Οθόνη Επιλογής Αντιπάλου

Η δενδρική δομή της οθόνης (σκηνής) επιλογής αντιπάλου αποτελείται από τρεις κόμβους:

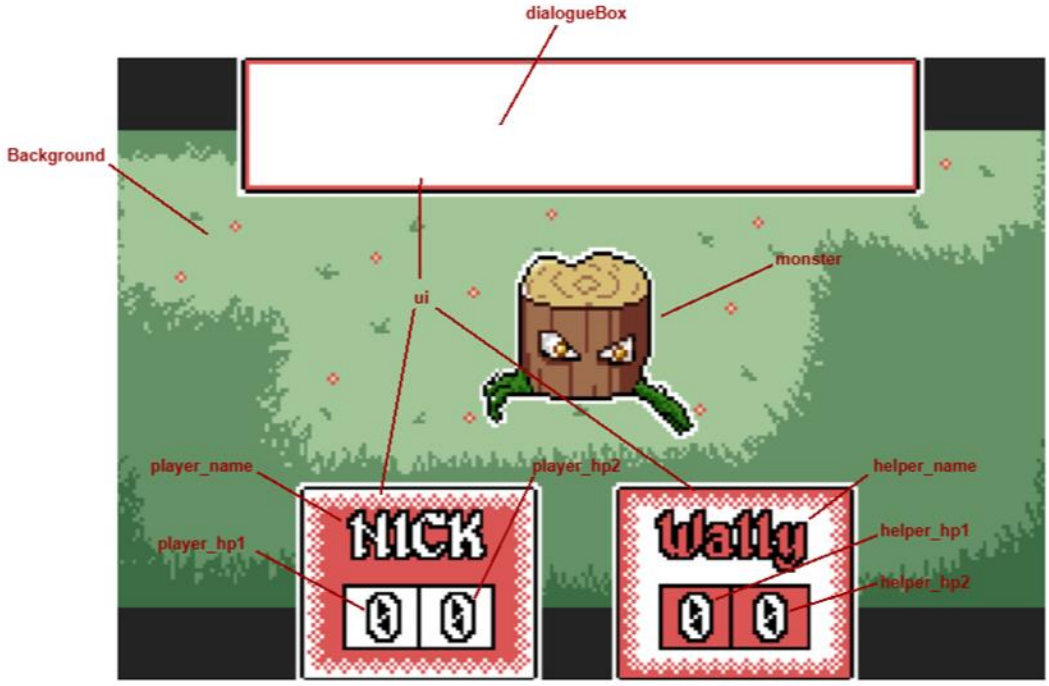
- Ο κόμβος-ρίζα (*enemy_selection*), στον οποίο είναι συνδεδεμένο ένα αρχείο κώδικα γραμμένο σε C#. Ο κώδικας αυτός διαχειρίζεται τις εισόδους του παίκτη (*user input*) και υλοποιεί την λογική της επιλογής.
- Ένας κόμβος τύπου *TextureRect* με όνομα *background*. Ο ρόλος του είναι να απεικονίζει το παρασκήνιο της οθόνης, η οποία περιλαμβάνει τους διαθέσιμους αντίπαλους του παιχνιδιού.
- Ένας δεύτερος κόμβος τύπου *TextureRect* με όνομα *enemy_selector*. Λειτουργεί ως δείκτης, τον οποίο μετακινεί ο παίκτης για να επιλέξει τον αντίπαλο που θέλει να αντιμετωπίσει.

```
public partial class enemy_selection: Node {

//Δήλωση του πεδίου που θα αποθηκεύσει την αναφορά στον κόμβο
//enemy_selector.

    public TextureRect enemy_selector;

//Ορισμός πεδίων τύπου Vector2 που αντιστοιχούν στις προκαθορισμένες
//συντεταγμένες των τεσσάρων αντιπάλων στην οθόνη.

    public Vector2 upLeft = new Vector2 (18,4);

    …

//Μέθοδος που καλείται αυτόματα από τη Godot μια φορά, όταν ο κόμβος είναι
έτοιμος.

public override void _Ready(){
```

```
  //Ανάκτηση της αναφοράς στον κόμβο enemy_selector από τη δενδρική δομή της
  //σκηνής.

      enemy_selector = GetNode<TextureRect> ("background/enemy_selector");

  //Αρχικοποίηση της τοποθεσίας του στην πάνω-αριστερή θέση.

      enemy_selector.Position = upLeft;

  }

//Μέθοδος που καλείται αυτόματα από τη Godot για κάθε γεγονός εισόδου
//(InputEvent).

public override void _Input (InputEvent @event){

//Συνθήκη που ικανοποιείται όταν ο παίκτης πατήσει το αριστερό πλήκτρο στο
//πληκτρολόγιο του.

    if (@event.IsActionPressed("ui_left")){

//Συνθήκη που ικανοποιείται αν ο κόμβος enemy_selector βρίσκεται στην
//τοποθεσία "πάνω και δεξιά"

            if(enemy_selector.Position == upRight){

//Η τοποθεσία του κόμβου αλλάζει στις συνταραγμένες που αναπαριστούν την
//τοποθεσία "πάνω και αριστερά"

                enemy_selector.Position = upLeft;

            }else if (enemy_selector.Position == downRight){

                enemy_selector.Position = downLeft;

            }

    }

    //Συνθήκη που ικανοποιείται αν ο παίκτης πατήσει το πλήκτρο "enter" στο
    //πληκτρολόγιο του.

        if (@event.IsActionPressed("ui_accept")){

    //Πρόσβαση στο GameManager, ένας "καθολικός διαχειριστής" που αποθηκεύει
    //πληροφορίες οι οποίες δεν χάνονται κατά την αλλαγή σκηνής.

        var gm = GameManager.Instance;

        if (enemy_selector.Position = upLeft){

    //Αποθήκευση των δεδομένων του αντιπάλου που βρίσκεται "πάνω και αριστερά"
    //στην οθόνη στον GameManager, ώστε να φορτωθούν στην επόμενη σκηνή.

            gm.Monster =1;

            gm.enemyName = "LOGMON'

            …

        }else if …

    //Κλήση της μεθόδου για τη μετάβαση στην επόμενη σκηνή (οθόνη μάχης).

        GoToNextScene();

}
```

Κώδικας 3.3: Διαχείριση της Οθόνης Επιλογής Αντιπάλου

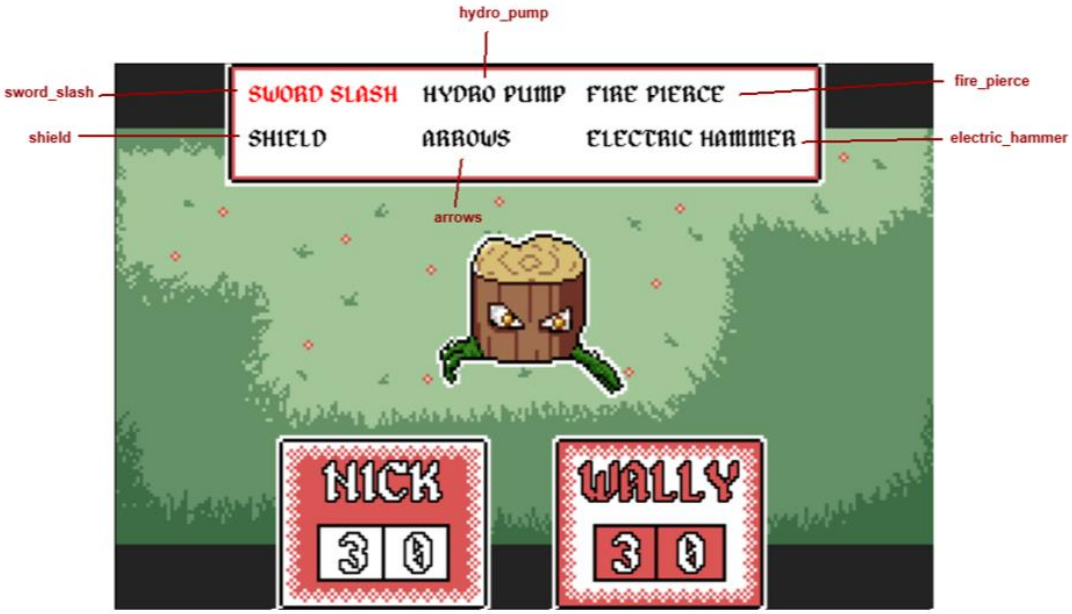
Το αρχείο κώδικα που είναι συνδεδεμένο με τον κόμβο *enemy_selection* υλοποιεί την κύρια λειτουργία του **μηχανισμού 1: "Διαμόρφωση Μάχης"**. Συγκεκριμένα, διαχειρίζεται την επιλογή αντιπάλου από τον παίκτη και αποθηκεύει τα αντίστοιχα χαρακτηριστικά για την αρχικοποίηση της οντότητας *Enemy*.



Εικόνα 3.6(α): Οθόνη Μάχης

Η δενδρική δομή της οθόνης (σκηνής) μάχης απαρτίζεται από τους εξής κόμβους:

- Τον κόμβο-ρίζα (*root*), στον οποίο είναι συνδεδεμένο ένα αρχείο κώδικα γραμμένο σε C#. Η κύρια λειτουργία του είναι η διαχείριση του μεγαλύτερου μέρους της λογικής του παιχνιδιού.
- Έναν κόμβο (*background*) τύπου *TextureRect*, ο οποίος αναπαριστά το παρασκήνιο της μάχης. Στο συγκεκριμένο παιχνίδι, το παρασκήνιο απεικονίζει τη χλωρίδα του μαγικού δάσους, όπου ταξιδεύουν οι χαρακτήρες.
- Έναν κόμβο (*ui*) τύπου *TextureRect*, ο οποίος χρησιμοποιείται για την αναπαράσταση των οντοτήτων *dialogueBox*, *playerBox* και *helperBox*.
- Έξι κόμβους (*sword_slash*, *shield*, *hydro_pump*, *arrows*, *fire_pierce*, *electric_hammer*) τύπου *Label*, των οποίων η βασική λειτουργία είναι να απεικονίζουν τις έξι διαθέσιμες ενέργειες του παίκτη στην οθόνη.
- Τέσσερις κόμβους τύπου *Label* που αξιοποιούνται για την απεικόνισή των αριθμητικών χαρακτηριστικών *playerBox.CHP* (*player_hp1*, *player_hp2*) και *helperBox.CHP* (*helper_hp1*, *helper_hp2*), καθώς και δύο επιπλέον κόμβους του ίδιου τύπου για τα χαρακτηριστικά *playerBox.Name* (*player_name*) και *helperBox.Name* (*helper_name*).
- Έναν κόμβο (*monster*) τύπου Sprite2D για την αναπαράσταση του επιλεγμένου αντιπάλου.
- Έναν κόμβο (*dialogue*) τύπου *Label*, ο οποίος είναι υπεύθυνος για την εμφάνιση του κειμένου που προβάλλει η οντότητα *dialogueBox*.



Εικόνας 3.6(β): Οθόνη Μάχης

Η διαδικασία αρχικοποίηση της οθόνης μάχης υλοποιείται στο αρχείο κώδικα του κόμβου "*root*". Εντός της μεθόδου *Ready()*, η οποία καλείται αυτόματα όταν ο κόμβος είναι έτοιμος, πραγματοποιείται η ανάκτηση των αναφορών για το σύνολο των κόμβων της ιεραρχίας της σκηνής, ώστε να είναι δυνατή η επεξεργασία τους. Στη συνέχεια, η μέθοδος *LoadFighter()* αναλαμβάνει την αρχικοποίηση των στατιστικών των χαρακτήρων, ενώ η *hpUi()* αναλαμβάνει την διαχείριση των μεταβολών των χαρακτηριστικών *playerBox.CHP* και *helperBox.CHP*.

```
public partial Root : Node {
    public Label dialogue;
    public Label sword_slash;
    …
    public character player;
    …
    public Texture2D Sprite1;
    …

public override void _Ready(){
    dialogue = GetNode<Label> ("background/ui/dialogue");
    …
//Αρχικοποίηση του dialogueBox με το εισαγωγικό μήνυμα της μάχης.
    dialogue.Text = "THE BATTLE BEGINS. PRESS ENTER!");
```

```
//Φόρτωση όλων των γραφικών των αντιπάλων.
    Sprite1 = GD.Load<Texture2D>("Absolute Path");
    …
    LoadFighters();
    hpUi (player.CHP, helper.CHP);
}
```

Κώδικας 3.4: Κώδικας Αρχικοποίησης της σκηνής μάχης

```
public void LoadFighters(){
//Πρόσβαση στον κεντρικό διαχειριστή (GameManager) για την ανάκτηση των
//καθολικών μεταβλητών της μάχης.
        var gm = GameManager.Instance;
        monsterId = gm.Monster;
//Δημιουργία τριών αντικειμένων τύπου character για τους τρεις χαρακτήρες και
//αρχικοποίηση των χαρακτηριστικών τους.
        enemy = new character (gm.enemyName, gm.monsterHp, …);
        player = new character (…, …);
        helper = new character (…, …);
//Αντιστοίχιση των κατάλληλων γραφικών στον κόμβο 'monster' βάση του του
//αναγνωριστικού του επιλεγμένου αντιπάλου.
        if (monsterId == 1){
        monster.Texture = Sprite1;
        }else if …
}
```

Κώδικας 3.5: Μέθοδος LoadFighters()

```
public void hpUi(){
//Αν η τιμή του player.CHP είναι μεγαλύτερη του 0, τα δύο ψηφία της μετατρέπονται
//σε strings και αποθηκεύονται στους κόμβους player_hp1 και player_hp2. Αν είναι 0
//ή μικρότερη τότε αυτόματα οι κόμβοι παίρνουν τιμή 0.
    if(player.CHP > 0) {
        player_hp1.Text = (player.CHP/10).ToString();
        player_hp2.Text = (player.CHP%10).ToString();
    }else{
        player_hp1.Text = "0";
        player_hp2.Text = "0";
    }
```

```
//Παρόμοια διαδικασία και για το χαρακτηριστικό helper.CHP.
        if (helper.CHP >0) {…
}
```

Κώδικας 3.6: Μέθοδος hpUi()

3.5.3. Κύριος Μηχανισμός

Για την υλοποίηση της κύριας λογικής που διαχειρίζεται τη ροή της μάχης, χρησιμοποιούνται τρία *enumerations* που αναπαριστούν τις οντότητες *State*, *turnManager* και *Losses*.

```
public enum State{
        NS = 0,        //No State
        MD = 1,        //Move Declaration
        AM = 2,        //Aftermath
        ST = 4         //Special State
}

public enum Losses{
        NL = 0,        //No Losses
        PL = 1,        //Player Lost
        EL = 2,        //Enemy Lost
        HL = 4         //Helper Lost
}
```

```
public enum Turn{
        PT = 0,        //Player Turn
        HT= 1,         //Helper Turn
        ET= 2          //Enemy Turn
}

//Αρχικοποίηση των χαρακτηριστικών turnManager.currentTurn,
//turnManager.previousTurn και των οντοτήτων State και Losses.

public Turn currentTurn = Turn.ET;

public Turn previousTurn = Turn.HT;

public State state = State.NS;

public Losses losses = Losses.NL;
```

Κώδικας 3.7: Κωδικοποίηση και αρχικοποίηση State, Losses και turnManager

Η εναλλαγή μεταξύ των τεσσάρων διακριτών φάσεων μιας σειράς υλοποιείται με μια δομή ελέγχου *if-else-if* εντός της μεθόδου *_Input* του αρχείου κώδικα *root*. Η δομή αυτή λειτουργεί ως μια μηχανή καταστάσεων, όπου η τρέχουσα τιμή της οντότητας *State* καθορίζει ποιο τμήμα της λογικής θα εκτελεστεί.

```
public override void _Input (InputEvent @event){
        if (@event.IsActionPressed("ui_accept")){
                if (state == State.NS){
                        …
                }else if (state == State.MD){
                        …
```

```
        }else if (state == State.AM){
                …
        }else if (state == State.ST){
                …
        }
    }
```

Κώδικας 3.8: Δομή Ελέγχου που εναλλάσει τις φάσεις μιας σειράς.

Κατά τη φάση της προετοιμασίας και επιλογής ενέργειας (*state = State.NS*), αρχικά πραγματοποιείται η ενημέρωση τιμών των χαρακτηριστικών *playerBox.CHP* και *helperBox.CHP* μέσω της μεθόδου *hpUi()*. Στη συνέχεια, η μέθοδος *turnHandler()* αναλαμβάνει τον καθορισμό του ενεργού χαρακτήρα (*μηχανισμός 4*) και την επιλογή της κίνησης του (*μηχανισμός 5*). Τέλος, γίνεται ενημέρωση της οντότητας *State* σε *MD*.

```
if (state == State.NS){
        hpUi(player.CHP, helper.CHP);
        TurnHandler();
        state = State.MD;
}
```

Κώδικας 3.9: Φάση προετοιμασίας και επιλογής ενέργειας.

```
Public void TurnHandler(){
        previousTurn = currentTurn;
        if (currentTurn == Turn.PT && losses == Losses.NL){
                currentTurn = Turn.HT;
                dialogue.Text = $"IT'S {helper.Name}'S TURN!";
//Εδώ, η λογική της τεχνητής νοημοσύνης καθορίζει την ενέργεια του βοηθού και
//αναθέτει την τιμή στη μεταβλητή chosenAction.
        }else if (currentTurn == Turn.HT && losses == Losses.NL || currentTurn ==
Turn.PT && losses == Losses.HL){
        currentTurn = Turn.ET;
        dialogue.Text = $"IT'S {enemy.Name}'S TURN!";
//Εδώ, η επιλογή της ενέργειας του εχθρού καθορίζεται από έναν αλγόριθμο
//παραγωγής τυχαίων αριθμών, ο οποίος αναθέτει την τιμή της μεταβλητής
//chosenAction.
        }else if (currentTurn == Turn.ET){
                currentTurn = Turn.PT;
```

```
//Για την σειρά του παίκτη, ο κόμβος dialogue γίνεται αόρατος και οι κόμβοι των
//διαθέσιμων ενεργειών γίνονται ορατοί. Το σύστημα αναμένει την επιλογή του
//χρήστη (επιβεβαιώνεται με το πλήκτρο "enter"), η οποία θα αποθηκεύσει το
//αναγνωριστικό της αντίστοιχης κίνησης στη μεταβλητή chosenAction.

                dialogue.Visible = false;

                sword_slash.Visible = true;

                …

        }

}
```

Κώδικας 3.10: Μέθοδος TurnHandler()

Κατά τη φάση της εκτέλεσης και υπολογισμού της ενέργειας (*State.MD*), αρχικά πραγματοποιείται έλεγχος για τον προσδιορισμό του ενεργού χαρακτήρα, ώστε να καθοριστεί η οντότητα *atkPattern* (*μηχανισμός 5*). Ακολούθως, δημιουργείται ένα στιγμιότυπο της κλάσης *move* (θα αναλυθεί στην επόμενη υπό-ενότητα). Το στιγμιότυπο αυτό αρχικοποιείται με δεδομένα από τις οντότητες *chosenAction*, *Player*, *Helper*, *Enemy*, *atkPattern* και *Counters*. Στη συνέχεια, καλείται η μέθοδος *executeAction()* του αρχικοποιημένου στιγμιότυπου η οποία υπολογίζει και εφαρμόζει τα αποτελέσματα της κίνησης στην κατάσταση του παιχνιδιού (*μηχανισμός 6)*.

Στη συνέχεια, ελέγχεται η σημαία *counterWasActive*, για την περίπτωση που η επιλεγμένη κίνηση επηρέασε στατιστικά τα οποία ήταν ήδη τροποποιημένα. Τέλος, ο έλεγχος αυτός καθορίζει εάν η διαδικασία θα επιστρέψει στην αρχική φάση (*State.NS*) ή εάν θα προχωρήσει στην επόμενη (*State.AM*) (*μηχανισμός 9*).

```
        else if (state == State.MD){
                if (currentTurn == Turn.PT){
                        moveInstance = new move(chosenAction, player, helper, enemy, 1,
                Counter);
                        moveInstance.executeAction();
//Οι επιλογές του παίκτη γίνονται πάλι αόρατο και ο κόμβος του διαλόγου γίνεται πάλι ορατός.
                sword_slash.Visible = false;
                …
                dialogue.Visible
                }else if (currentTurn = Turn.HT){
//Η συνάρτηση moveChooser() είναι ο αλγόριθμος που αναλύεται στο επόμενο κεφάλαιο για την
//πρόβλεψη της βέλτιστης κίνησης.
                        chosenAction = moveChooser(player, helper, enemy, Counter, monderId);
                        moveInstance = new move(chosenAction, player, helper, enemy, 2,
                Counter);
                        moveInstance.executeAction();
                }else{…}
```

```
            if (moveInstance.counterWasActive == false){
//Ο παίκτη ενημερώνεται για την κίνηση που μόλις εκτελέστηκε.
                    dialogue.Text = moveInstance.amDialogue;
                    state = State.AM;
            }else{
//Ο παίκτης ενημερώνεται για την αποτυχία εκτέλεσης της κίνησης και την επανάληψη της σειράς.
                    dialogue.Text = moveInstance.mdDialogue;
                    state = State.NS;
                    currentTurn = previousTurn;
            }
        }
```

Κώδικας 3.11: Φάση Υπολογισμού και Εκτέλεσης Ενέργειας

Στη φάση της επίλυσης και ενημέρωσης (*State.AM*), η μέθοδος *counterHandler()* ενημερώνει όλους τους ενεργούς μετρητές του παιχνιδιού και εκτελεί τους απαραίτητους ελέγχους για τον εντοπισμό εκπλήρωσης κάποιας συνθήκης που μπορεί να οδηγήσει το παιχνίδι σε ειδική κατάσταση(*μηχανισμός 7*). Ταυτόχρονα, ενημερώνει τον παίκτη μέσω του κόμβου *dialogue* για τα αποτελέσματα της κίνησης που μόλις εκτελέστηκε.

```
else if (state == State.AM){
        counterHandler();
        dialogue.Text = moveInstance.amDialogue;
}
```

Κώδικας 3.12: Φάση Επίλυσης και Ενημέρωσης

```
//Δήλωση και αρχικοποίηση ενός πίνακα που αναπαριστά τους 5 μετρητές του
//παιχνιδιού.
public int [] counter = {0, 0, 0, 0, 0};

public void counterHandler(){
//Αύξηση του μετρητή κατά 1 αν η τιμή του είναι στο εύρος [1-3] και εκμηδένιση του
//αν είναι ίση με 4, καθώς και μεταβολή του στατιστικού που μεταβλήθηκε από την
//κίνηση που παρακολουθεί στην αρχική της τιμή.
        if (currentTurn == Turn.PT){
                if (counter[0] > 0 && counter[0] < 4){
                        counter[0] = counter[0] + 1;
                else if (counter[0] == 4){
                        counter[0] = 0;
```

```
                player.DEF = player.DEF -3;

            }

        }else if (currentTurn == Turn.HT) …

//Ελεγχος εκπλήρωσης της συνθήκης που συνεχίζει την διαδικασία στην επόμενη
//σειρά και των συνθηκών που την οδηγεί σε μια ειδική κατάσταση.

        If (player.CHP > 0 && helper.CHP >0 && enemy.CHP > 0 || player.CHP >0
&& enemy.CHP && losses == Losses.HL){

            state = State.NS;

        else if (player.CHP < 1){

            state = State.ST;

            losses = Losses.PL

        else if…

}
```

Κώδικας 3.13: Μέθοδος counterHandler()

Τέλος, στην φάση διαχείρισης ειδικών καταστάσεων, το σύστημα εξετάζει τη συγκεκριμένη συνθήκη στην οποία έχει εισέλθει το παιχνίδι. Ανάλογα με τη συνθήκη αυτή, είτε γίνεται τερματισμός της μάχης, είτε πραγματοποιείται οποιαδήποτε άλλη απαραίτητη τροποποίηση.

```
if (losses == Losses.PL){

        dialogue.Text = $"AFTER A HEROIC BATTLE {player.Name} LOST.
BETTER LUCK NEXT TIME";

//Εντολή που κλείνει το παιχνίδι.

        GetTree().Quit();

}else if (losses == Losses.HL){

        Dialogue.Text = $"{helper.Name} TRIED THEIR BEST BUT THE BATTLE
WAS TOO FIERCE FOR THEM";

        state = State.NS;

}else if (losses = Losses.EL){

        dialogue.Text = $"YOUR PERSEVERANCE PAID OFF. YOU WIN!";

//Μέθοδος που οδηγεί το σύστημα στην οθόνη επιλογής αντιπάλου.

        GoToEnemySelection();

}
```

Κώδικας 3.14: Φάση Διαχείρισης Ειδικών Καταστάσεων

3.5.4. Μηχανισμοί Εκτέλεσης Κινήσεων

Η εκτέλεση των κινήσεων υλοποιείται μέσω μιας κλάσης με το όνομα *move*. Κάθε φορά που ένας χαρακτήρας εκτελεί μια ενέργεια, δημιουργείται ένα στιγμιότυπο αυτής της κλάσης, το οποίο είναι υπεύθυνο για την εφαρμογή των αντίστοιχων αποτελεσμάτων στην κατάσταση του παιχνιδιού.

Η διαδικασία αυτή ξεκινάει από τον κατασκευαστή της κλάσης, ο οποίος αρχικοποιεί τις ιδιότητες του αντικειμένου με τις τιμές που λαμβάνει ως παραμέτρους από τη κεντρική κλάση *root*. Στην συνέχεια, ο κώδικας στην κεντρική λογική του παιχνιδιού καλεί την μέθοδο *executeAction*(), που είναι υπεύθυνη για την εκτέλεση της κίνησης. Η μέθοδος αυτή αρχικά προσδιορίζει τον τύπο της επίθεσης και χρησιμοποιώντας την οντότητα *atkPattern* που έχει ήδη οριστεί, καθορίζει τους ρόλους του επιτιθέμενου και του αμυνόμενου. Τέλος, ανάλογα με τον τύπο της κίνησης καλείται η μέθοδος *LoadMove()* για να φορτώσει τα δεδομένα της και η κατάλληλη μέθοδος υπολογισμού (*attackCalculations()*, *hpCalculations()* ή *statCalculations()*) για την εφαρμογή της λογικής της.

```
public partial class move: Node{
//Δήλωση των ιδιωτικών πεδίων της κλάσης.
        private int _chosenAction
        …
//Δήλωση των δημόσιων ιδιοτήτων της κλάσης.
        public int moveNumber{
                get { return _chosenAction; }
                set { _chosenAction = value; }
        }
        …
//Δήλωση πεδίων για την προσωρινή αποθήκευση των χαρακτηριστικών της τρέχουσας κίνησης.
        private int movePower,…
        …
//Δημιουργία στιγμιότυπου της κλάσης Random για την παραγωγή τυχαίων αριθμών.
        private Random rng = new Random();
//Κατασκευαστής της κλάσης που αρχικοποιεί ένα νέο αντικείμενο "move" με τις παραμέτρους που
δέχεται κατά τη δημιουργία του από την κλάση "root".
        public move(int chosen_action, …){
                moveNumber = chosen_action;
                …
        }
```

```
//Μέθοδος που εκτελεί την κεντρική λογική της κλάσης.
	public void executeAction(){
		int moveType = moveNumber/100;
//Ορισμός του επιτιθέμενου και του αμυνόμενου από το μοτίβο επίθεσης.
		switch(attackPattern){
			case 1:
				Attacker = Helper;
				Defender = Enemy;
				break;
			case 2: …
		}
//Εντοπισμός του τύπου της κίνησης και κλήση των αντίστοιχων μεθόδων για φόρτωση δεδομένων και
//υπολογισμό αποτελεσμάτων.
		if (moveType == 1){
			LoadMove(1, "File Path για το JSON αρχείο με τις επιθέσεις");
			attackCalculations();
		}else{
			LoadMove(2, "File Path για το JSON αρχείο με τις κινήσεις
		στατιστικών/πόντων ζωής");
			if (movePower == 0){
				hpCalculations();
			}else{
				statCalculations();
			}
		}
	}
```

Κώδικας 3.15: Κατασκευαστής και μέθοδος executeAction() της κλάσης move

Η μέθοδος *LoadMove()* ανακτά τα δεδομένα της κίνησης από το αντίστοιχο αρχείο JSON και τα αναθέτει σε πεδία του στιγμιότυπου.

```
	public void LoadMove(int moveType, string fileName){

		string jsonString = "";
```

```
//Πραγματοποιείται πρόσβαση στο αρχείο που ορίζεται από τη μεταβλητή 'fileName' σε κατάσταση
//ανάγνωσης (read-only). Η χρήση της εντολής using διασφαλίζει το κλείσιμο του αρχείου μετά
//την ανάγνωση. Το περιεχόμενο του αποθηκεύεται ως μια συμβολοσειρά στη μεταβλητή
//'jsonString'.
                using(var file = FileAccess.Open(fileName, FileAccess.ModeFlags.Read)){
                        jsonString = file.GetAsText();
                }
//Η συμβολοσειρά jsonString αποθηκεύεται ως μια δομή δεδομένων τύπου JsonNode με όνομα root.
                JsonNode root = JsonNode.Parse(jsonString);
//Γίνεται έλεγχος αν το περιεχόμενο της μεταβλητής root είναι JSON πίνακας.
                if(root is JsonArray movesArray){
//Ένας βρόχος που κάνει προσπέλαση όλα τα στοιχεία του πίνακα. Σε κάθε επανάληψη αποθηκεύει το
τρέχον στοιχείο προσωρινά την μεταβλητή moveObject.
                        foreach(var moveObject in movesArray){
//Εντός του βρόχου, για κάθε στοιχείο, επιβεβαιώνεται ότι είναι αντικείμενο JSON και το
αντικείμενο ανατίθεται στη μεταβλητή 'currentMove'.
                                if(moveObject is JsonObject currentMove){
//Η κεντρική συνθήκη αναζήτησης. Γίνονται 2 ελέγχει: (1) Ο πρώτος προσπαθεί να βρει την
//ιδιότητας "Number". Αν δεν υπάρχει, επιστρέφει false. (2) Αν η ιδιότητα βρεθεί, η τιμή //της
συγκρίνεται με το αναγνωριστικό της κίνησης που αναζητούμε.
                                        if(currentMove.TryGetPropertyValue("Number", out JsonNode
                                numberNode) && numberNode.GetValue<int>() == moveNumber){
//Ανάλογα τον τύπο της κίνησης φορτώνονται τα κατάλληλα δεδομένα στα πεδία του στιγμιότυπου.
                                                if(moveType == 1){
                                                        moveName =
                                                moveObject["Name"]?.GetValue<string>() ?? "";
                                                        movePower =
                                                moveObject["Power"]?.GetValue<int>() ?? 0;
                                                        …
                                                }else{…}
}}}}
```

Κώδικας 3.16: Μέθοδος LoadMove()

Η μέθοδος *attackCalculations()* είναι υπεύθυνη για τον υπολογισμό της τελικής ζημιάς. Αρχικά, καλεί άλλες μεθόδους για να υπολογίσει τους τροποποιητές της επίθεσης και στη συνέχεια την αρχική συνολική ζημιά. Ακολούθως, ελέγχει το αναγνωριστικό της

κίνησης για να εφαρμόσει ειδικές τροποποιήσεις όπου απαιτείται, δημιουργεί τα κείμενα που θα ενημερώσουν τον παίκτη για την κίνηση και το αποτέλεσμα της και τέλος, αφαιρεί την τελική ζημιά από τους τρέχοντες πόντους ζωής του αμυνόμενου.

```
        public void attackCalculations(){
//Κλήση μεθόδων για τον υπολογισμό τροποποιητών ζημιάς και αποθήκευση τους σε μεταβλητές.
                int spdMod = speedComparison();
                float vulResMod = vulResCalc();
//Κλήση μεθόδου για τον υπολογισμό της συνολικής ζημιάς.
                int moveDmg = atkPow(spdMod, vulResMod);
//Υλοποίηση της ειδικής συμπεριφοράς των κινήσεων 111,113,114
                if (moveNumber == 111 || moveNumber == 113 || moveNumber == 114){
                …
                }
//Δημιουργία του κείμενου που ενημερώνει τον παίκτη για την κίνηση που εκτελέστηκε.
                mdDialogue = $"{Attacker.Name} {dialogue1} {Defender.Name} {dialogue2}";

//Υλοποίηση της ειδικής συμπεριφοράς της κίνησης 103 και προσαρμογή του κειμένου mdDialogue.
                if(moveNumber == 103){
                        int arrows = rng.Next(1,4);
                        moveDmg = moveDmg*arrows;
                        mdDialogue = $"{Attacker.Name} {dialogue1} {Defender.Name} {arrows}
                {dialogue2}";
                }
//Αφαίρεση της συνολικής ζημιάς από τους τρέχοντες πόντους ζωής του αμυνόμενου, δημιουργία
//του κείμενου που ενημερώνει τον παίκτη για τα αποτελέσματα της κίνησης και απενεργοποίηση
//της σημαίας counterWasActive.
                Defender.CHP = Defender.CHP - moveDmg;
                amDialogue = $"{Defender.Name} TOOK {moveDmg} POINTS OF DAMAGE!";
                counterWasActive = false;
        }
```

Κώδικας 3.17: Μέθοδος attackCalculations()

Οι μέθοδοι *speedComparison()* και *vulResMod()* υλοποιούν τους *μηχανισμούς 6.1.α* και *6.2.β* αντίστοιχα, ενώ η μέθοδος *atkPow()* υλοποιεί τους *μηχανισμούς 6.1.γ*, *6.1.δ* επιστρέφοντας την τιμή της συνολικής ζημιάς στην μεταβλητή *moveDmg*.

```
private int speedComparison(){
        if (Attacker.SPD > (Defender.SPD + moveDelay)){
                return 1;
        else if…
}
```

Κώδικας 3.18: Μέθοδος speedComparison()

```
 private float vulResCalc(){
//Μετατροπή των λιστών αδυναμιών και αντοχών του αμυνόμενου και των τύπων της επίθεσης σε
//strings ώστε να συγκριθούν.
                string vul = Defender.VUL.ToString();
                string res = Defender.RES.ToString();
                string atk = moveDmgType.ToString();
                …
//Εύρεση των κοινών τύπων μεταξύ της επίθεσης και των αδυναμιών του αμυνόμενου.
                for (x=0; x<atkLen; x++){
                        for(y=0; y<vulLen; y++){
                                if(vul[y] == atk[x]){
                                        atkVul = atkVul +1;
                }}}
//Παρόμοια για τις αντοχές.
                …
//Αφαίρεση του αριθμού των αδυναμιών από τον αριθμό των αντοχών.
                atkRes = atkRes - atkVul;
                switch (atkRes){
                        case 2:
                                return 0.25f;
                        case 1: …
 }
```

Κώδικας 3.19: Μέθοδος vulResCalc()

```
 private int atkPow(int spdMod, float vulResMod){
                int atkPow, moveDmgCategory=0;
                string atk = moveDmgType.ToString();
                bool special=false, physical=false;
```

```
//Ελέγχονται όλοι οι τύποι επίθεσης για την κατηγορία στην οποία ανήκουν ώστε να προσδιοριστεί
η κατηγορία της επίθεσης συνολικά.
            for (x==0; x< atk.Length; x++){
                  if((Regex.IsMatch(atk[x], "[1-6]") && special == false){
                        moveDmgCategory = moveDmgCategory + 1;
                        special true;
                  }else if ((Regex.IsMatch(atk[x], "[7-9]") && physical == false){
                        moveDmgCategory = moveDmgCategory + 2;
                        physical = true;
            }}
            if (moveDmgCategory == 2){
                  atkPow = (int)Math.Ceiling((Attacker.ATK-0.5*Defender.DEF)+ movePower);
            else if (moveDmgCategory == 1){
                  atkPow = (int)Math.Ceiling(Attacker.SP_ATK-0.5*Defender.SP_DEF) +
            movePower);
            else if (moveDmgCategory == 3){
                  atkPow = (int)Math.Ceiling(0.5*(Attacker.ATK-0.5*Defender.DEF)+
            0.5*(Attacker.SP_ATK-0.5*Defender.SP_DEF) + movePower);
            }
            atkPow = (int)Math.Ceiling((atkPow*vulResMod + 0.25(atkPow*spdMod))*0.5);
            return atkPow;
      }
```

Κώδικας 3.20: Μέθοδος atkPow()

Η μέθοδος *hpCalculations()* εκτελεί τους απαραίτητους υπολογισμούς για την εφαρμογή των πόντων ζωής στον αντίστοιχο χαρακτήρα (*μηχανισμός 6.3*).

```
      public void hpCalculations(){
            counterWasActive = false;
            moveHP = (rng.Next(1,4))*2;

            if (moveTarget == "Player"){
                  Player.CHP = player.CHP + moveHP;

                  if (Player.CHP > Player.HP){
                        moveHP = Player.HP - Player.CHP + moveHP;
```

```
                    Player.CHP = Player.HP;
                }
                mdDialogue = $"{Helper.Name} {dialogue1} {Player.Name}";
                amDialogue = $"{Player.Name} {dialogue2} {moveHP} HP!";
            else if (moveTarget == "Helper") …
//Παρόμοια διαδικασία για τις κινήσεις επαναφοράς πόντων ζωής του βοηθού και του αντιπάλου.
    }
```

Κώδικας 3.21: Μέθοδος hpCalculations()

Τέλος, η μέθοδος *statCalculations()* είναι υπεύθυνη για την υλοποίηση των κινήσεων αλλαγής στατιστικών. Αρχικά αναγνωρίζει την κίνησης προς εκτέλεση, ελέγχει τον μετρητή που την παρακολουθεί και τέλος εφαρμόζει τις αντίστοιχες μεταβολές στα στατιστικά του χαρακτήρα αν αυτό είναι επιτρεπτό.

```
    public void statCalculations(){
        string stat0 = "";
        string stat1 = "";

        character Target = null;
        If (moveTarget == "Player"){
            Target = Player;
        else if…
//Η μεταβλητή moveStat χωρίζεται σε 2 strings σε περίπτωση που είναι 2 τα στατιστικά που
//πρέπει να μεταβληθούν.
        string[] stats = moveStat.Split(' ');
        stat0 = stats[0];
        if (Counter[moveCounter] == 0){
//Λήψη της ιδιότητας μέσω Reflection, με βάση το όνομα που περιέχεται στο string 'stat0'.
            PropertyInfo propertyInfo = typeof(character).GetProperty(stat0);
//Ανάκτηση της τρέχουσας τιμής της ιδιότητας από το αντικείμενο 'Target'.
            int currentValue = (int)propertyInfo.GetValue(Target);
//Υπολογισμός της νέας τιμής και ανάθεσης της πίσω στην ιδιότητα του αντικειμένου 'Target'.
            propertyInfo.SetValue(Target, currentValue + movePower);
            Counter[moveCounter] = 1;
            counterWasActive = false;
//Επανάληψη της διαδικασίας σε περίπτωση που η μεταβλητή moveStat έχει δύο στατιστικά.
```

```
            if (stats.Length == 2){
                ...
            }
        else{
            counterWasActive = true;
        }

        if (counterWasActive == true){
            mdDialogue = dialogue3;
        else{
            mdDialogue = $"{Attacker.Name} {dialogue1}";
            amDialogue = $"{Target.Name} {dialogue2}";
        }
    }
```

Κώδικας 3.22: Μέθοδος statCalculations()

3.6. Ορισμός των Σταθερών Δεδομένων

Η πλήρης υλοποίησης των μηχανισμών του πρωτοτύπου σε κώδικα επιτρέπει τον υπολογισμό του συνόλου των μεταβλητών του. Οι μεταβλητές αυτές, ωστόσο, προαπαιτούν ένα σύνολο σταθερών τιμών για την ορθή λειτουργία τους. Η παρούσα ενότητα επικεντρώνεται στον προσδιορισμό αυτών των τιμών.

3.6.1. Προσδιορισμός των Στατιστικών του Παίκτη και του Βοηθού

Οι χαρακτήρες του παίκτη και του βοηθού αποτελούν δύο από τις κεντρικές οντότητες γύρω από τις οποίες λειτουργεί το σύστημα της μάχης. Ως βάση για τον καθορισμό του εύρους τιμών των στατιστικών τους, αποτέλεσε το παιχνίδι *Dungeons & Dragons*. Στο συγκεκριμένο παιχνίδι, οι **ικανότητες (*abilities*)** ενός χαρακτήρα λαμβάνουν τιμές από 3 έως 20, με την τιμή 10 να αναπαριστά ένα μέσο επίπεδο ικανότητας [20].

Συνεπώς, οι χαρακτήρες του παίκτη και του βοηθού θα έχουν τιμές που θα περιστρέφονται γύρω από αυτό τον μέσο όρο (10). Η συγκεκριμένη σχεδιαστική επιλογή σκοπεύει στην μεγαλύτερη ευελιξία κατά τον ορισμό των στατιστικών των αντιπάλων. Η χρήση μέσων τιμών επιτρέπει στον παίκτη και τον βοηθό να αναπτύξουν ευέλικτες στρατηγικές απέναντι όλων των τύπων εχθρών.

Όσον αφορά τις αδυναμίες και τις αντοχές τους, τα χαρακτηριστικά αυτά παραμένουν κενά, ώστε να μην υπάρχει πλεονέκτημα ή μειονέκτημα για κανέναν αντίπαλο. Τέλος, ο παίκτης, ως μαχητής, διαθέτει ενισχυμένα στατιστικά που αφορούν φυσικές επιθέσεις, ενώ ο βοηθός, ως μάγος, έχει αυξημένα τα αντίστοιχα στατιστικά ειδικών επιθέσεων.

Παίκτης

Όνομα	Τιμή
HP	30
CHP	30
ATK	12
DEF	12
SP.ATK	10
SP.DEF	10
SPD	10
VUL	-
RES	-

Βοηθός

Όνομα	Τιμή
HP	30
CHP	30
ATK	10
DEF	10
SP.ATK	12
SP.DEF	12
SPD	10
VUL	-
RES	-

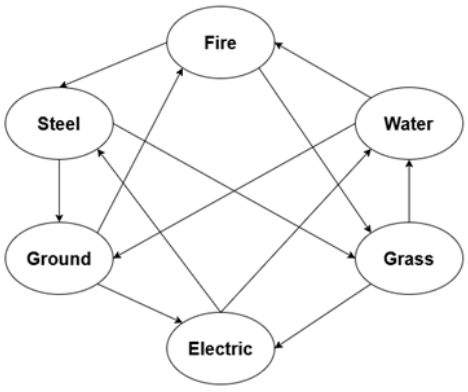
Εικόνα 3.7: Στατιστικά Παίκτη και Βοηθού

Εξαίρεση αποτελούν τα χαρακτηριστικά *HP* και *CHP*, τα οποία μπορούν να λάβουν τιμές πάνω του ορίου.

3.6.2 Προσδιορισμός Τιμών για τις Κινήσεις του Παίκτη και του Βοηθού

Για τον προσδιορισμό των τιμών των κινήσεων του παίκτη και του βοηθού θα αξιοποιηθεί η αρχή των αμετάβατων σχέσεων. Αρχικά, θα οριστούν οι σχέσεις αντοχής και αδυναμίας μεταξύ των τύπων ζημιάς του παιχνιδιού.

Special Types				
Num	**Name**	**Vul**	**Res**	**NR**
1	Fire	2,5	3,6	1,4
2	Water	3,4	1,5	2,6
3	Grass	1,6	2,4	3,5
4	Electric	3,5	2,6	1,4
5	Ground	2,6	1,4	3,5
6	Steel	1,4	3,5	2,6



Εικόνα 3.8: Οι σχέσεις μεταξύ των ειδικών τύπων

Pysical Types	
Num	**Name**
7	Bludgeoning
8	Piercing
9	Slashing

Εικόνα 3.9: Οι φυσικοί τύποι του παιχνιδιού

Στην συνέχεια, θα ανατεθεί ένας τύπος σε καθέναν από τους τέσσερις αντιπάλους, καθορίζοντας έτσι τις τιμές των χαρακτηριστικών *Character.VUL* και *Character.RES*. Επιπλέον, στις τιμές αυτές θα προστεθούν και φυσικοί τύποι ζημιάς, οι οποίοι είναι ανεξάρτητοι από τον κύριο τύπο του αντιπάλου.




Type	Grass	**Type**	Ground	**Type**	Water	**Type**	Fire
VUL	1,6,7	**VUL**	2,6	**VUL**	3,4,7	**VUL**	2,5,8
RES	2,4	**RES**	1,4,7,8	**RES**	1,5,8	**RES**	3,6,9

Εικόνα 3.10: Αδυναμίες και Αντοχές των Αντιπάλων

Τέλος, πραγματοποιείται σύγκριση των τύπων ζημιάς κάθε κίνησης με τις αντοχές και αδυναμίες των τεσσάρων αντιπάλων. Αποτέλεσμα αυτής της διαδικασίας είναι η δημιουργία ενός πίνακα (*εικόνα 3.11*) που απεικονίζει την αποτελεσματικότητα κάθε κίνησης, αναδεικνύοντας αυτές με το μεγαλύτερο πλεονέκτημα ή μειονέκτημα.

Για τον καθορισμό του χαρακτηριστικού *movePower* κάθε κίνησης, υπολογίζεται ένας συνολικός τροποποιητής αντοχών/αδυναμιών. Ο τροποποιητής αυτός προκύπτει από τον πολλαπλασιασμό των τιμών του *vulResMod* για κάθε κίνησης έναντί όλων των αντιπάλων.

Η τιμή του *movePower* καθορίζεται από τον συνολικό τροποποιητή ως εξής:

- Τροποποιητής x0.25: *movePower* = 6.
- Τροποποιητής x0.5:*movePower* = 5.
- Τροποποιητής x1: *movePower* = 4.
- Τροποποιητής x2: *movePower* = 2.

Εξαίρεση αποτελεί η κίνηση "*Arrows*", καθώς η τελική της ζημιά μπορεί να διπλασιαστεί ή να τριπλασιαστεί κατά τον υπολογισμό της, οπότε της δίνεται η τιμή 1.

Το χαρακτηριστικό *moveDelay* δέχεται την τιμή ένα για κινήσεις που έχουν *movePower* κάτω από πέντε και δύο για τις υπόλοιπες.

Name	dmgTypes	Op1	Op2	Op3	Op4	Total	movePower	moveDelay
Sword Slash	7	x2	x0.5	x2	x1	x2	2	1
Hydro Pump	2,9	x0.5	x2	x1	x1	x1	4	1
Arrows	6,8	x2	x1	x0.5	x1	x1	1	1
Fire Pierce	1,8	x2	x0.25	x0.25	x2	x0.25	6	2
Electric Hammer	4,9	x0.5	x0.5	x2	x0.5	x0.25	6	2
Fire Bolt	1	x2	x0.5	x0.5	x1	x0.5	5	2
Thunderbolt	4	x0.5	x0.5	x2	x1	x0.5	5	2
Rock Smash	5,9	x1	x1	x0.5	x1	x0.5	5	2
Thorns	3,8	x1	x0.5	x1	x1	x0.5	5	2

Εικόνα 3.11: Πίνακας σύγκρισης αντοχών/αδυναμιών των κινήσεων με τους αντίπαλους.

3.6.3. Προσδιορισμός των Στατιστικών και των Κινήσεων των Αντιπάλων

Το παιχνίδι περιλαμβάνει τέσσερις αντιπάλους. Ο καθορισμός των στατιστικών τους θα βασιστεί στην αρχή της Ορθογώνιας Διαφοροποίησης Μονάδων, που αναλύθηκε στην *ενότητα 2.4*. Συγκεκριμένα, οι τέσσερις αντίπαλοι θα διαθέτουν στατιστικά που τους επιτρέπουν να ακολουθούν τις παρακάτω στρατηγικές:

- **Αντίπαλος 1**: Ο αντίπαλος αυτός ακολουθεί μια στρατηγική ανάλογη με αυτή του παίκτη, διαθέτοντας ισορροπημένα στατιστικά που κυμαίνονται γύρω από τον μέσο όρο.
- **Αντίπαλος 2:** Ο αντίπαλος αυτός ακολουθεί μια αμυντική στρατηγική. Χαρακτηρίζεται από υψηλά αμυντικά στατιστικά και χαμηλά επιθετικά, με στόχο την απορρόφηση ζημιάς και την επικράτηση μέσω αντοχής.
- **Αντίπαλος 3:** Ο αντίπαλος αυτός ακολουθεί μια επιθετική στρατηγική. Διαθέτοντας υψηλή επίθεση και χαμηλή άμυνα, επικεντρώνεται στην πρόκληση μεγάλης ζημιάς, με στόχο να νικήσει τους αντίπαλος πριν ο ίδιος ηττηθεί.
- **Αντίπαλος 4:** Ο αντίπαλος αυτός ακολουθεί μια στρατηγική συνεχής φθοράς και επιβίωσης. Αντί να βασίζεται σε υψηλή άμυνα, αξιοποιεί κινήσεις αλλαγής στατιστικών και επαναφοράς πόντων ζωής, με σκοπό την καθυστέρηση της μάχης και την επικράτηση μέσω της σταδιακής εξάντλησης των αντιπάλων.

Ο προσδιορισμός των τιμών που επιτρέπουν την υλοποίηση των παραπάνω στρατηγικών μπορεί να αντιμετωπιστεί ως ένα πρόβλημα ικανοποίησης περιορισμών. Το πρόβλημα αυτό μπορεί να επιλυθεί μέσω της εφαρμογής ενός αλγορίθμου εξαντλητικής αναζήτησης.

Οι ζητούμενες τιμές σε κάθε περίπτωση είναι τα απροσδιόριστα στατιστικά των αντιπάλων (*HP*, *ATK*, *DEF*, *SP.ATK*, *SP.DEF*, *SPD*) και οι παράμετροι της βασικής τους επίθεσης (*movePower*, *moveDelay*). Η εύρεση τους πραγματοποιείται ακολουθώντας την εξής διαδικασία:

1. Ορισμός του χώρου λύσεων, μέσω του καθορισμού του εύρους τιμών για κάθε εμπλεκόμενη μεταβλητή.
2. Καθορισμός των περιορισμών και αξιολόγηση κάθε υποψήφιας λύσης ως προς την τήρηση τους.

3. Επιλογή της τελικής λύσης από το σύνολο όσων ικανοποιούν τους περιορισμούς. Στην παρούσα υλοποίηση, ως τελική λύση ορίζεται η αποδεκτή λύση που έχει την μικρότερη απόκλιση από τον μέσο όρο (*mean*) όλων των έγκυρων λύσεων.

Αντίπαλος 1: Ισορροπημένη στρατηγική

Για την εύρεση των στατιστικών του αντίπαλου 1 ορίζονται τα εξής πεδία μεταβλητών:

- HP: 70
- ATK : [8-14]
- DEF: [8-14]
- SP.ATK: [8-14]
- SP.DEF: [8-14]
- SPD: 11
- movePower: [3-8]
- moveDelay: 1

Τα στατιστικά *SPD* και *moveDelay* ορίζονται στις τιμές 11 και 1, λόγω της ειδικής κίνησης στατιστικών που διαθέτει ο αντίπαλος. Η κίνηση αυτή αυξάνει το *SPD* του κατά 3, διασφαλίζοντας έτσι ότι οι επιθέσεις του αποκτούν πλεονέκτημα ταχύτητας μόνο μετά τη χρήση της.

Ο ορισμός των περιορισμών του προβλήματος προϋποθέτει τη χρήση ενός συνόλου ενδιάμεσων μεταβλητών:

- **plDmg:** Η μέγιστη ζημιά που προκαλεί ο παίκτης στον αντίπαλο με την πιο αποτελεσματική του κίνηση.
- **heDmg:** Η μέγιστη ζημιά που προκαλεί ο βοηθός στον αντίπαλο με την πιο αποτελεσματική του κίνηση.
- **enDmgPl:** Η μέγιστη ζημιά που προκαλεί ο αντίπαλος στον παίκτη με την βασική του κίνηση.
- **enDmgHe:** Η μέγιστη ζημιά που προκαλεί ο αντίπαλος στον βοηθό με την βασική του κίνηση.
- **Turns0:** Ο αριθμός σειρών που απαιτούνται ώστε ο παίκτης να εξαντλήσει τους πόντους ζωής του αντιπάλου, χρησιμοποιώντας την πιο αποτελεσματική του κίνηση.
- **Turns1:** Ο αριθμός σειρών που απαιτούνται ώστε ο βοηθός να εξαντλήσει τους πόντους ζωής του αντιπάλου, χρησιμοποιώντας την πιο αποτελεσματική του κίνηση.
- **Turns2:** Ο αριθμός σειρών που απαιτούνται ώστε ο αντίπαλος να εξαντλήσει τους πόντους ζωής του παίκτη, χρησιμοποιώντας τη βασική του κίνηση.
- **Turns3:** Ο αριθμός σειρών που απαιτούνται ώστε ο αντίπαλος να εξαντλήσει τους πόντους ζωής του βοηθού, χρησιμοποιώντας τη βασική του κίνηση.
- **Turns4:** Ο αριθμός σειρών που απαιτούνται ώστε ο αντίπαλος να μειώσει τους πόντους ζωής είτε του παίκτη είτε του βοηθού κάτω από το μισό, επιτιθέμενος εναλλάξ.
- **min:** Η τιμή του χαμηλότερου στατιστικού του αντιπάλου.
- **max:** Η τιμή του υψηλότερου στατιστικού του αντιπάλου.
- **av:** Ο μέσος όρος των μεταβλητών Turns0 και Turns1.

Οι τιμές των μεταβλητών *plDmg*, *heDmg*, *enDmgPl* και *enDmgHe* υπολογίζονται μέσω προσομοίωσης του μηχανισμού ζημιάς, εξετάζοντας όλους τους συνδυασμούς για τον εντοπισμό των μέγιστων τιμών. Οι μεταβλητές *Turns0-Turns4* προκύπτουν από την αφαίρεση της αντίστοιχης ζημιάς ανά σειρά από τους πόντους ζωής του στόχου, αντικατοπτρίζοντας τον αριθμό των επιθέσεων που απαιτούνται για τον μηδενισμό (*Turns0-3*) ή την μείωση κάτω από το μισό (για *Turns4*).

Βάσει των μεταβλητών που έχουν οριστεί, μια υποψήφια λύση πρέπει να ικανοποιεί τους ακόλουθους περιορισμούς για να θεωρηθεί αποδεκτή:

- **Περιορισμός 1:** (*Turns2* < *Turns0*) && (*Turns2* < *Turns1*) &&(*Turns3* < *Turns0*) && (*Turns3* < *Turns1*)
- **Περιορισμός 2:** *av* == (*turns4* +1)
- **Περιορισμός 3:** (*max-min*) <= 3
- **Περιορισμός 4:** Για κάθε τιμή του στατιστικού *ATK*, γίνεται αποδεκτή μόνο η πρώτη λύση που ικανοποιεί τους παραπάνω περιορισμούς. Όλες οι επόμενες υποψήφιες λύσεις με την ίδια τιμή *ATK* απορρίπτονται.

Με αυτούς τους περιορισμούς προκύπτουν τα παρακάτω τελικές λύσεις:

HP	ATK	DEF	SP.ATK	SP.DEF	SPD	movePower	moveDelay
70	9	12	9	12	11	5	1
70	10	9	12	9	11	7	1
70	11	8	11	8	11	7	1
70	12	9	10	9	11	7	1
70	13	11	10	13	11	3	1
70	14	11	11	13	11	3	1

Εικόνα 3.12: Αποδεκτές τιμές για τα στατιστικά του αντιπάλου 1.

Ο μέσος όρος (mean) αυτών των λύσεων είναι:

- DEF: 9/11
- SP.ATK: 10/11
- SP.DEF: 9/13
- movePower: 7

Κατόπιν σύγκρισης, η πέμπτη λύση είναι αυτή που προσεγγίζει περισσότερο το υπολογισμένο μέσο όρο. Συνεπώς, τα στατιστικά της επιλέγονται για τον πρώτο αντίπαλο.

<u>Αντίπαλος 2: Αμυντική Στρατηγική</u>

Για την εύρεση των στατιστικών του αντίπαλου 2 ορίζονται τα εξής πεδία μεταβλητών:

- HP: 100
- ATK : [5-10]
- DEF: [15-17]
- SP.ATK: [5-10]
- SP.DEF: [15-17]
- SPD: [8-11]
- movePower: [3-6]
- moveDelay: [1-2]

Για τον ορισμό των περιορισμών θα χρησιμοποιηθούν οι ίδιες ενδιάμεσες μεταβλητές:

- **Περιορισμός 1:** (*Turns2* < *Turns0*) && (*Turns2* < *Turns1*) &&(*Turns3* < *Turns0*) && (*Turns3* < *Turns1*)
- **Περιορισμός 2:** *av* == (*turns4* +3)

- **Περιορισμός 3:** Η διαφορά μεταξύ των στατιστικών DEF και SP.DEF δεν πρέπει να υπερβαίνει το 2.
- **Περιορισμός 4:** Για κάθε τιμή του στατιστικού *DEF*, γίνεται αποδεκτή μόνο η πρώτη λύση που ικανοποιεί τους παραπάνω περιορισμούς. Όλες οι επόμενες υποψήφιες λύσεις με την ίδια τιμή *DEF* απορρίπτονται.

Με αυτούς τους περιορισμούς προκύπτουν τα παρακάτω τελικές λύσεις:

HP	ATK	DEF	SP.ATK	SP.DEF	SPD	movePower	moveDelay
100	5	14	5	14	8	6	1
100	5	15	5	14	8	6	1
100	5	16	5	14	8	6	1
100	5	17	5	14	8	6	1

Εικόνα 3.13: Αποδεκτές τιμές για τα στατιστικά του αντιπάλου 2.

Οι λύσεις με εξαίρεση το στατιστικό DEF είναι πανομοιότυπες. Ενδεικτικά επιλέγεται η πρώτη λύση ως η τελική.

<u>Αντίπαλος 3: Επιθετική Στρατηγική</u>

Για την εύρεση των στατιστικών του αντίπαλου 3 ορίζονται τα εξής πεδία μεταβλητών:

- HP: 50
- ATK : [15-19]
- DEF: [8-10]
- SP.ATK: [15-19]
- SP.DEF: [8-10]
- SPD: [8-12]
- movePower: [5-10]
- moveDelay: [1-2]

Για τον ορισμό των περιορισμών θα χρησιμοποιηθούν οι ίδιες ενδιάμεσες μεταβλητές:

- **Περιορισμός 1:** (*Turns2* < *Turns0*) && (*Turns2* < *Turns1*) &&(*Turns3* < *Turns0*) && (*Turns3* < *Turns1*)
- **Περιορισμός 2:** *av* == (*turns4* +1)
- **Περιορισμός 3:** Η διαφορά μεταξύ των στατιστικών ATK και SP.ATK δεν πρέπει να υπερβαίνει το 2.
- **Περιορισμός 4:** Για κάθε τιμή του στατιστικού *ATK*, γίνεται αποδεκτή μόνο η πρώτη λύση που ικανοποιεί τους παραπάνω περιορισμούς. Όλες οι επόμενες υποψήφιες λύσεις με την ίδια τιμή *ATK* απορρίπτονται.
- **Περιορισμός 5:** (*av*>*turns2*) && (*av* > *turns3*)

Με αυτούς τους περιορισμούς προκύπτουν τα παρακάτω αποδεκτές λύσεις:

HP	ATK	DEF	SP.ATK	SP.DEF	SPD	movePower	moveDelay
50	15	10	15	10	10	5	1
50	16	10	15	10	10	5	1
50	17	10	15	10	10	5	1

Εικόνα 3.14: Αποδεκτές τιμές για τα στατιστικά του αντιπάλου 3.

Οι τελικές τιμές είναι πανομοιότυπες με εξαίρεση το στατιστικό ATK, οπότε επιλέγεται ενδεικτικά η δεύτερη για τον καθορισμό των τελικών τιμών.

<u>Αντίπαλος 4: Στρατηγική Εξασθένισης και Επιβίωσης</u>

Για την εύρεση των στατιστικών του αντίπαλου 2 ορίζονται τα εξής πεδία μεταβλητών:

- HP: 50
- ATK : [8-15]
- DEF: [8-15]
- SP.ATK: [8-15]
- SP.DEF: [8-15]
- SPD: [8-12]
- movePower: [4-8]
- moveDelay: [1-2]

Για τον ορισμό των περιορισμών θα χρησιμοποιηθούν οι ίδιες ενδιάμεσες μεταβλητές:

- **Περιορισμός 1:** (*Turns2* < *Turns0*) && (*Turns2* < *Turns1*) &&(*Turns3* < *Turns0*) && (*Turns3* < *Turns1*)
- **Περιορισμός 2:** *av* == (*turns4* +1)
- **Περιορισμός 3:** (max-min) <= 5
- **Περιορισμός 4:** Για κάθε τιμή του στατιστικού *SP.ATK*, γίνεται αποδεκτή μόνο η πρώτη λύση που ικανοποιεί τους παραπάνω περιορισμούς. Όλες οι επόμενες υποψήφιες λύσεις με την ίδια τιμή *SP.ATK* απορρίπτονται.

Με αυτούς τους περιορισμούς προκύπτουν τα παρακάτω αποδεκτές λύσεις:

HP	ATK	DEF	SP.ATK	SP.DEF	SPD	movePower	moveDelay
50	10	15	10	13	11	7	1
50	10	15	11	13	11	6	1
50	10	15	12	13	10	8	1
50	10	15	13	13	10	7	1
50	10	15	14	13	10	6	1
50	10	15	15	13	10	5	1

Εικόνα 3.15: Αποδεκτές τιμές για τα στατιστικά του αντιπάλου 4.

Ο μέσος όρος (mean) αυτών των λύσεων είναι:

- ATK: 10
- DEF: 15

- SP.DEF: 13
- SPD: 10
- movePower: 6/7
- moveDelay: 1

Παρόλο που οι λύσεις είναι σε μεγάλο βαθμό πανομοιότυπες, η τέταρτη και η πέμπτη λύση είναι οι μοναδικές που αντικατοπτρίζουν πλήρως το μέσο όρο. Ενδεικτικά θα επιλεχτεί η τέταρτη λύση ως η τελική.

Βάσει του παραπάνω αλγορίθμου, καθορίζονται τα στατιστικά των αντιπάλων, καθώς και αυτά της βασικής τους κίνησης. Οι αντίπαλοι 1-3, ωστόσο, διαθέτουν και μια δεύτερη επίθεση, η οποία παρουσιάζει μια ιδιαίτερη συμπεριφορά. Κατά την εκτέλεση της επίθεσης αυτής, ένας αλγόριθμος παραγωγής τυχαίων αριθμών παράγει έναν αριθμό (1 ή 2). Εάν ο αριθμός αυτός είναι 1, τότε η συνολική ζημιά της επίθεσης υποδιπλασιάζεται, ενώ σε αντίθετη περίπτωση διπλασιάζεται. Σε αυτές τις κινήσεις αποδίδεται τιμή *movePower* τέτοια, ώστε η τελική τους ζημιά να είναι μικρότερη από εκείνη της βασικής επίθεσης στην περίπτωση του υποδιπλασιασμού, και μεγαλύτερη στην περίπτωση του διπλασιασμού.

Name	movePower	moveDelay	Damage
Vines	5	1	6
Whip	2	1	4/7
Stone Paunch	6	1	3
Magnitude	6	1	2/4
Watergun	7	1	7
Tackle	5	1	5/8

Εικόνα 3.16: Σύγκριση των βασικών και δευτερεύοντων επιθέσεων των αντιπάλων 1-3.

Ακολουθώντας τη μεθοδολογία του *Adams*, σύμφωνα με την οποία η ανάπτυξη παιχνιδιών βασίζεται στον κύκλο "Κατασκευή, Αξιολόγηση, Επανάληψη", ως αποτέλεσμα της αξιολόγησης των στατιστικών των κινήσεων των αντιπάλων στο παιχνίδι, πραγματοποιείται αλλαγή της τιμής *movePower* της κίνησης *Watergun* από 5 σε 7. Η προσαρμογή αυτή στοχεύει στην καλύτερη εφαρμογή της επιθετικής στρατηγικής του αντιπάλου 3 στο παιχνίδι. Επιπλέον, η κίνηση *Magnitude* θα ενισχύεται κατά μια μονάδα στη συνολική της ζημιά μετά τον διπλασιασμό, ώστε να έχει συνολική ζημιά ίση με 4.

Με την ολοκλήρωση των παραπάνω τροποποιήσεων, τα στατιστικά των αντιπάλων και των κινήσεων τους είναι πλήρως διαμορφωμένα.

HP	ATK	DEF	SP.ATK	SP.DEF	SPD	RES	VUL
70	13	11	10	13	11	2,4	1,6,7
100	5	14	5	14	8	1,4,7,8	2,6
50	16	10	15	10	10	1,5,8	3,4,7
50	10	15	13	13	10	3,6,9	2,5,8

Name	movePower	moveDelay	moveDmgTypes
Vines	5	1	3,8
Whip	2	1	7
Stone Paunch	6	1	5,9
Magnitude	6	1	5
Tackle	5	1	9
Watergun	7	1	2,9
Flames	7	1	1

Εικόνα 3.17: Τα στατιστικά των αντιπάλων και των επιθέσεων τους.

3.6.4. Καθορισμός Τιμών για τον Αλγόριθμο Παραγωγής Τυχαίων Πόντων Ζωής

Το παιχνίδι ενσωματώνει τρεις κινήσεις επαναφοράς πόντων ζωής, η αποτελεσματικότητα των οποίων καθορίζεται από έναν αλγόριθμο παραγωγής τυχαίων αριθμών. Για τις κινήσεις που επαναφέρουν τους πόντους ζωής του παίκτη και του βοηθού, ορίζονται τρεις τιμές, ανάλογα με το αποτέλεσμα του αλγορίθμου:

- **Αποτέλεσμα (3):** Αναπλήρωση 6 πόντων ζωής.
- **Αποτέλεσμα (2):** Αναπλήρωση 4 πόντων ζωής.
- **Αποτέλεσμα (1):** Αναπλήρωση 2 πόντων ζωής.

Η επιλογή των συγκεκριμένων τιμών έγινε με κριτήριο τη μέση ζημιά που προκαλούν οι επιθέσεις των αντιπάλων που είναι ίση με 5.

Αντίστοιχα, για την κίνηση επαναφοράς πόντων ζωής του αντιπάλου, οι τιμές διαμορφώνονται ως εξής:

- **Αποτέλεσμα (3):** Αναπλήρωση 12 πόντων ζωής.
- **Αποτέλεσμα (2):** Αναπλήρωση 8 πόντων ζωής.
- **Αποτέλεσμα (1):** Αναπλήρωση 4 πόντων ζωής.

Ο καθορισμός των τιμών αυτόν βασίστηκε στο γεγονός ότι το άθροισμα των πιο αποτελεσματικών επιθέσεων του παίκτη και του βοηθού εναντίον του αντιπάλου που διαθέτει την κίνηση είναι ίσο με 14.

Κεφάλαιο 4- Αλγόριθμος Πρόβλεψης Κινήσεων

Στο *κεφάλαιο 3*, ο προσδιορισμός της οντότητας *chosenAction* για τον χαρακτήρα του παίκτη πραγματοποιείται μέσω της διεπαφής χρήστη, ενώ στην περίπτωση του αντιπάλου μέσω ενός αλγορίθμου παραγωγής τυχαίων αριθμών. Η μοναδική περίπτωση στην οποία δεν προσδιορίζεται ο τρόπος επιλογής κίνησης για έναν χαρακτήρα είναι αυτή του βοηθού, όπου ο τρόπος συμπλήρωσης της οντότητας *chosenAction* επεξηγείται με την αναφορά σε έναν αφηρημένο αλγόριθμο επιλογής κίνησης. Στο παρόν κεφάλαιο, θα αναλυθεί η λειτουργία του εν λόγω αλγορίθμου. Ειδικότερα, στην *Ενότητα 4.1*, θα εξεταστεί το πρόβλημα το οποίο επιλύει ο αλγόριθμος και θα οριστεί η λειτουργία του. Στις *Ενότητες 4.2-4.3*, θα αναλυθούν τρία στοιχεία που συμβάλλουν στη λειτουργία του αλγορίθμου και τέλος, στην *Ενότητα 4.4*, θα πραγματοποιηθεί η υλοποίηση του αλγορίθμου σε κώδικα, με σκοπό την ενσωμάτωση του στον κώδικα του πρωτοτύπου.

4.1. Ορισμός του Αλγορίθμου Πρόβλεψης Κινήσεων

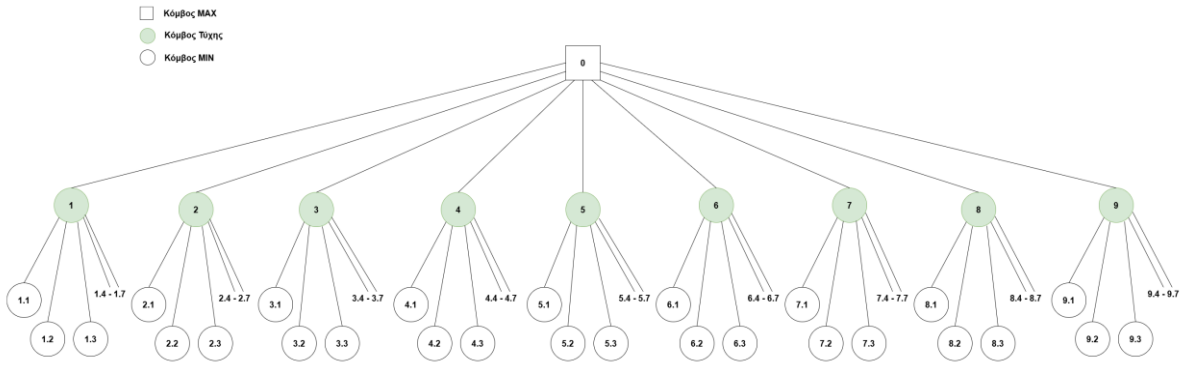
Η επιλογή της βέλτιστης κίνησης που μπορεί να εκτελέσει ο βοηθός στο παιχνίδι αποτελεί ένα πρόβλημα αναζήτησης. Για τον λόγο αυτό, ορίζεται ένας χώρος καταστάσεων, τα στοιχεία του οποίου αναπαριστούν τις διαφορετικές καταστάσεις στις οποίες μπορεί να βρεθεί το παιχνίδι. Κάθε κατάσταση προσδιορίζεται από το σύνολο των τιμών των παρακάτω μεταβλητών:

- Οι τρέχοντες πόντοι ζωής των χαρακτήρων (*Player.CHP*, *Helper.CHP*, *Enemy.CHP*)
- Οι τιμές των μετρητών που παρακολουθούν τις κινήσεις αλλαγής στατιστικών (*Counters*).

Η αρχική κατάσταση της αναζήτησης είναι η τρέχουσα κατάσταση του παιχνιδιού κατά την έναρξη της σειράς του βοηθού. Σε αυτό το πρόβλημα, δεν υπάρχει μια προκαθορισμένη κατάσταση-στόχος, αλλά ο σκοπός είναι ο εντοπισμός της κίνησης που οδηγεί στην καλύτερη κατάσταση, όπως αυτή αποτιμάται από μια συνάρτηση αξιολόγησης. Οι ενέργειες που προκαλούν μετάβαση από μια κατάσταση σε άλλη αντιστοιχούν στις διαθέσιμες κινήσεις του βοηθού και του αντιπάλου.

Ο αλγόριθμος αναζήτησης αναπτύσσει ένα δένδρο παιχνιδιού βάθους 2, το οποίο αναπαριστά όλα τα πιθανά αποτελέσματα μετά από μια κίνηση του βοηθού και την ακόλουθη κίνηση του αντιπάλου. Στη συνέχεια, ο αλγόριθμος *Expeciminimax* διατρέχει το δένδρο, αποτιμά τις τελικές καταστάσεις (κόμβους-φύλλα) μέσω της συνάρτησης αξιολόγησης και υπολογίζει την αναμενόμενη τιμή κάθε αρχικής κίνησης, επιλέγοντας τελικά αυτή με τη μέγιστη τιμή.

Η *εικόνα 4.1* απεικονίζει τη δομή του δένδρου παιχνιδιού που αναπτύσσει ο αλγόριθμος για την επιλογή της βέλτιστης κίνησης έναντι των αντιπάλων 1-3.



Εικόνα 4.1: Το δένδρο παιχνιδιού του Αλγορίθμου

Οι εννέα κόμβοι στο πρώτο επίπεδο (βάθος 1) αντιστοιχούν στις καταστάσεις-αποτελέσματα των εννέα διαθέσιμων κινήσεων του βοηθού: *Firebolt*, *Thunderbolt*, Rock *Smash*, *Thorns*, *Encouragement Spell*, *Intimidation Spell* , *Distraction, Heal Player* και *Heal Self*.

Κάθε ένας από αυτούς τους κόμβους τύχης έχει διάδοχους που αναπαριστούν τα αποτελέσματα των πιθανών κινήσεων του αντιπάλου. Παρότι ο αντίπαλος διαθέτει τρεις κινήσεις (βασική επίθεση, δευτερεύουσα επίθεση, κίνηση αλλαγής στατιστικών), στο δένδρο απεικονίζονται επτά διάδοχοι. Αυτό συμβαίνει διότι η ζημιά της δευτερεύουσας επίθεσης τροποποιείται από έναν αλγόριθμο παραγωγής τυχαίων αριθμών, με αποτέλεσμα να δημιουργούνται δύο διαφορετικές πιθανές καταστάσεις για τη συγκεκριμένη κίνηση. Επιπλέον, ο αντίπαλος μπορεί να επιτεθεί είτε στον παίκτη, είτε στον βοηθό, διπλασιάζοντας έτσι τις πιθανές καταστάσεις που προκύπτουν από επιθέσεις από τρεις σε

έξι. Η έβδομη κατάσταση είναι αυτή που προκύπτει από την κίνηση αλλαγής στατιστικών που διαθέτει ο αντίπαλος, η οποία δεν έχει στόχο ούτε τον παίκτη, ούτε τον αντίπαλο.

Για τον αντίπαλο 4, αναπτύσσεται ένα παρόμοιο δένδρο παιχνιδιού, με τη διαφορά ότι κάθε κόμβος τύχης έχει τέσσερις διαδόχους αντί για επτά. Αυτό συμβαίνει επειδή ο συγκεκριμένος αντίπαλος δεν διαθέτει δευτερεύουσα επίθεση, αλλά μια κίνηση επαναφοράς πόντων ζωής, η οποία οδηγεί σε μια μόνο κατάσταση.

Ο αλγόριθμος για την επιλογή της βέλτιστης κίνησης παρουσιάζεται παρακάτω σε μορφή ψευδοκώδικα:

```
//Ανάκτηση των αρχικών δεδομένων από την τρέχουσα κατάσταση του παιχνιδιού.
Αρχικοποίηση CHP_Characters, Counters[], chosenEnemy, helperMoves[], enemyMoves[]
//Δήλωση δομής για την αποθήκευση της βαθμολογίας κάθε κίνησης
Δομή moveScore: (index, finalGrade)
//Δήλωση μεταβλητών που χρησιμοποιούνται από τον αλγόριθμο.
Δήλωση i,k,j,flag,grade,Number,sum,maxIndex,State0,State1,State2
//Δήλωση μιας λίστας που αποθηκεύει αντικείμενα τύπου moveScore
Δήλωση moveScores
//Αποθήκευση της αρχικής κατάστασης του παιχνιδιού (ρίζα) στη μεταβλητή State0
State0 = CHP_Characters, Counters
//Κύριος βρόγχος που εξετάζει κάθε μια από τις 9 πιθανές κινήσεις του βοηθού.
Για (i=0) επανάλαβε όσο (i<9):
//Έλεγχος δυνατότητας εκτέλεσης των κινήσεων αλλαγής στατιστικών. Αν ο μετρητής
μιας κίνησης είναι ήδη ενεργός, η κίνηση παραλείπεται.
        Αν (i>3 & i<7):
                flag = checkCounter(i, state0.Counters)
                Αν (flag = "Αληθής"):
                        Συνέχισε στην επόμενη επανάληψη
//Υπολογισμός της νέας κατάστασης (State1) μετά την εκτέλεση της κίνησης 'i' του
βοηθού
        State1 = helperAction(State0, chosenEnemy, helperMoves[], i)
        Number = 0
        Sum = 0
//Ορισμός του πλήθος κινήσεων του αντιπάλου.
        Αν (chosenEnemy >0 & chosenEnemy <4):
                k = 7
        Αλλιώς αν (chosenEnemy = 4):
                k = 4
//Εσωτερικός βρόχος που εξετάζει κάθε πιθανή κίνηση 'j' του αντιπάλου.
        Για (j=0) επανάλαβε όσο (j<k):
```

```
//Παρόμοιος έλεγχος και για την κίνηση αλλαγής στατιστικών του αντιπάλου
                Αν (j = 0):
                        flag = checkCounter(0)
                        Αν (flag = “Αληθής”):
                                Συνέχισε στην επόμενη επανάληψη
                State2 = enemyAction(State1, chosenEnemy, enemyMoves[], j)
                Grade = Evaluation(State2)
                Number = Number +1
                Sum = Sum + Grade
//Υπολογισμός της αναμενόμενης τιμής της κίνησης ‘i’ και αποθήκευση της στην λίστα.
        ExpectedValue = Sum /Number
        Πρόσθεσε νέο moveScore(index: i, finalGrade: Expected Value) στη λίστα
moveScores
//Ταξινόμηση της λίστας κινήσεων με φθίνουσα σειρά με βάση την βαθμολογία
(finalGrade).
Ταξινόμησε τη λίστα moveScores
//Έλεγχος αν υπάρχουν κινήσεις που έχουν αξιολογηθεί.
Αν (η λίστα moveScores δεν είναι κενή):
//Από προεπιλογή η πρώτη θέση της λίστας θεωρείται ότι έχει το στοιχείο με την
//μεγαλύτερη βαθμολογία.
        maxIndex = moveScores[0].index
//Έλεγχος ισοβαθμίας καλύτερων επιλογών, αν υπάρχουν 2 ή 3 κινήσεις που έχουν την
//ίδια καλύτερη βαθμολογία επιλέγεται τυχαία μια από αυτές.
        Αν( (moveScores.Πλήθος)>2 KAI moveScores[0].finalGrade ==
moveScores[2].finalGrade):
                maxIndex = moveScores[Τυχαίος αριθμός μεταξύ 0-2].index
        Αλλιώς αν( (moveScores.Πλήθος)>1 KAI moveScores[0].finalGrade ==
        moveScores[1].finalGrade):
                maxIndex = moveScores[Τυχαίος αριθμός μεταξύ 0-1].index
//Επιστροφή του αναγνωριστικού της βέλτιστης κίνησης.
        Επέστρεψε helperMoves[maxIndex]
Αλλιώς:
        Επέστρεψε helperMoves[0]
```

Αλγόριθμος 4.1: Αλγόριθμος Επιλογής Βέλτιστης Κίνησης.

4.2. Ορισμός και Τρόπος Εφαρμογής των Κινήσεων στις Καταστάσεις

Στην παρούσα ενότητα αναλύονται οι συναρτήσεις *helperAction()* και *enemyAction()*, εξετάζοντας στον τρόπο με τον οποίο τροποποιούν μια κατάσταση για να παράγουν τη κατάσταση-διάδοχο της.

Όπως έχει ήδη αναφερθεί, στο παιχνίδι υπάρχουν τρεις τύποι κινήσεων: επιθέσεις, αλλαγής στατιστικών, επαναφοράς πόντων ζωής.

Αρχικά, όσον αφορά τις κινήσεις αλλαγής στατιστικών, οι συναρτήσεις αυτές θέτουν την τιμή του μετρητή που τους αντιστοιχεί σε 1, σηματοδοτώντας με αυτό τον τρόπο την ενεργοποίηση τους.

Για τις κινήσεις επαναφοράς πόντων ζωής, γίνεται πρόσθεση των αντίστοιχων πόντων ζωής της κάθε κίνησης στον χαρακτήρα-στόχο.

Η συνάρτηση *helperAction()* διαχειρίζεται δύο από τις τρεις κινήσεις επαναφοράς πόντων ζωής: "*Heal Player*" και "*Heal Self*". Στην πρώτη περίπτωση, αυξάνει την τιμή της μεταβλητής *Player.CHP* κατά 4, ενώ στη δεύτερη, η αντίστοιχη αύξηση πραγματοποιείται στη μεταβλητή *Helper.CHP*. Αντίστοιχα, η συνάρτηση *enemyAction()* διαχειρίζεται την τρίτη κίνηση επαναφοράς πόντων ζωής: "*Heal Enemy*". Η αύξηση σε αυτή την περίπτωση γίνεται στην μεταβλητή *Enemy.CHP* και είναι ίση με 8.

Τέλος, οι επιθέσεις απαιτούν μια πιο σύνθετη διαδικασία υλοποίησης. Η συνάρτηση *helperAction()* υλοποιεί τέσσερις τέτοιες κινήσεις, ενώ η συνάρτηση *enemyAction()* επτά. Η πολυπλοκότητα αυτή οφείλεται στο ότι για τον υπολογισμό της τελικής ζημιάς και συνεπώς, τον προσδιορισμό της διάδοχης κατάστασης, πρέπει να ληφθούν υπόψη οι ακόλουθοι παράγοντες:

- Η ταυτότητα της κίνησης που εκτελείται, καθώς κάθε μια διαθέτει ξεχωριστά στατιστικά που καθορίζουν τη τελική ζημιά της.
- Ο στόχος της επίθεσης, καθώς τα μοναδικά στατιστικά κάθε χαρακτήρα τον καθιστούν περισσότερο ευάλωτο ή ανθεκτικό σε μια κίνηση.
- Η κατάσταση των μετρητών που αντιστοιχούν στις κινήσεις αλλαγής στατιστικών, καθώς τα τροποποιημένα στατιστικά ενός χαρακτήρα μπορεί να επηρεάσουν τον αριθμό της τελικής ζημιάς.

Για τον υπολογισμό της τελικής ζημιάς, λαμβάνοντας υπόψη όλους τους παραπάνω παράγοντες, αξιοποιείται ένας πίνακας με την όνομα **"Κατάλογος Ζημιών" (*dmgCatalogue*)**. Ο κατάλογος αυτός αποτελείται από ζεύγη κλειδιού-τιμής, όπου το κλειδί λειτουργεί ως αναγνωριστικό και η τιμή αντιστοιχεί στη ζημιά.

Στην *εικόνα 4.2* παρουσιάζεται η δομή των αναγνωριστικών:

attackPattern	moveNumber	counterState	possibility
XX	XXX	XXXXX	X

Εικόνα 4.2: Δομή αναγνωριστικών του Καταλόγου Ζημιών

Το αναγνωριστικό κωδικοποιεί το σύνολο των συνθηκών που καθορίζουν μια συγκεκριμένη τιμή ζημιάς στο παιχνίδι. Αποτελείται από τα ακόλουθα πεδία:

- **attackPattern:** Το πεδίο αυτό προσδιορίζει τον επιτιθέμενο και τον αμυνόμενο. Η κωδικοποίηση των χαρακτήρων είναι η εξής: P για τον παίκτη, H για τον βοηθό και οι αριθμοί 1-4 για τους αντίστοιχους αντιπάλους. Για παράδειγμα, η τιμή 1P

αναφέρεται σε μια επίθεση του αντιπάλου 1 προς τον παίκτη, ενώ η τιμή Η3 σε επίθεση του βοηθού προς τον αντίπαλο 3.

- **moveNumber:** Είναι το μοναδικό αναγνωριστικό της κίνησης που εκτελείται.
- **counterState:** Το πεδίο αυτό αποτυπώνει την κατάσταση όλων των μετρητών του παιχνιδιού κατά την στιγμή της επίθεσης. Πρόκειται για μια ακολουθία ψηφίων όπου κάθε θέση αντιστοιχεί σε έναν συγκεκριμένο μετρητή. Για παράδειγμα, η τιμή 00001 υποδηλώνει ότι οι πρώτοι τέσσερις μετρητές είναι ανενεργοί και μόνο ο πέμπτος (που αντιστοιχεί στην κίνηση αλλαγής στατιστικών του αντιπάλου) είναι ενεργός.
- **possibility:** Το πεδίο αυτό χρησιμοποιείται για να διακρίνει τις πιθανές τιμές ζημιάς των δευτερευόντων επιθέσεων που διαθέτουν οι αντίπαλοι. Το αποτέλεσμα αυτών των επιθέσεων καθορίζεται από έναν αλγόριθμο παραγωγής τυχαίων αριθμών που μπορεί να οδηγήσει σε δύο διαφορετικές τιμές ζημιάς. Το πεδίο αυτό δηλώνει ποια από τις δύο τιμές εξετάζει (1, για τη μειωμένη, 2 για την αυξημένη). Για τις κινήσεις που έχουν μόνο μια συγκεκριμένη τιμή ζημιάς, το πεδίο λαμβάνει πάντα την τιμή 1.

Βάσει αυτού του συστήματος, οι συναρτήσεις *helperAction()* και *enemyAction()* συνθέτουν το αναγνωριστικό που αντιστοιχεί στην τρέχουσα επίθεση και κατάσταση του παιχνιδιού. Στη συνέχεια, πραγματοποιούν αναζήτηση στον κατάλογο με αυτό το κλειδί, προκειμένου να ανακτήσουν την τιμή της ζημιάς που θα αφαιρεθεί από τους πόντους ζωής του στόχου.

Η πλήρης συμπλήρωση του καταλόγου με όλους τους πιθανούς συνδυασμούς είναι ένα υπολογιστικά χρονοβόρο εγχείρημα. Για τον λόγο αυτό, ο κατάλογος αρχικοποιείται μόνο με τις τιμές ζημιάς που αντιστοιχούν στην βασική κατάσταση του παιχνιδιού, δηλαδή όταν όλοι οι μετρητές είναι ανενεργοί. Επιπλέον, παραλείπονται οι τιμές ζημιάς των επιθέσεων του παίκτη καθώς δεν εμπεριέχονται στους υπολογισμούς του αλγορίθμου.

Εάν ο αλγόριθμος αναζητήσει ένα αναγνωριστικό που αντιστοιχεί σε ενεργοποιημένους μετρητές και δεν το εντοπίσει, τότε χρησιμοποιεί τη τιμή της ζημιάς που έχει η βασική κατάσταση (με ανενεργούς μετρητές). Μετά την εκτέλεση της κίνησης στο παιχνίδι, η πραγματική τιμή της ζημιάς καταχωρείται στον κατάλογο με το πλήρες αναγνωριστικό της, ώστε να χρησιμοποιηθεί σε μελλοντικές αναζητήσεις.

Για παράδειγμα, εάν πρέπει να υπολογιστεί η ζημιά της επίθεσης *Thunderbolt* (107) του βοηθού στον αντίπαλο 2, ενώ είναι ενεργός ο μετρητής της κίνησης "*Encouragement Spell*", ο αλγόριθμος θα αναζητήσει το αναγνωριστικό H2-107-01000-1. Αν η καταχώριση αυτή δεν υπάρχει, ο αλγόριθμος θα χρησιμοποιήσει τη ζημιά που αντιστοιχεί στο αναγνωριστικό H2-107-00000-1 για να δημιουργήσει τον διάδοχο κόμβο. Στη συνέχεια, μετά την εκτέλεση της κίνησης, η πραγματική ζημιά που προέκυψε θα καταχωρηθεί στον κατάλογο με το πλήρες αναγνωριστικό της.

Ακολουθεί η αναπαράσταση των συναρτήσεων *helperAction()*, *enemyAction()* και *checkCounter()* σε μορφή ψευδοκώδικα:

```
//Μεταβλητές για την κατασκευή του αναγνωριστικού της ζημιάς.

Δήλωση attackPattern, moveNumber, counterState, possibility, damageId, heal, dmg


//Διαχείριση των επιθέσεων του βοηθού.

Αν (i>=0 & i<4):

//Κατασκευή του αναγνωριστικού (damageId).

        attackPattern = "H" + chosenEnemy.ToString()
```

```
        moveNumber = helperMoves[i].toString()

        counterState = Counters[0].toString() + Counters[1].ToString() +
        Counters[2].ToString() + Counters[3].ToString() + Counters[4].ToString()

        possibility = "1"

        damageId = attackPattern + moveNumber + counterStates + possibility

//Ανάκτηση της τιμής ζημιάς από τον κατάλογο ζημιάς με βάση το αναγνωριστικό.

        dmg = SearchDmgCatalogue(damageId)

        Enemy.CHP = Enemy.CHP – dmg

//Αποθηκεύουμε τον αριθμό ζημιάς της επίθεσης ώστε να χρησιμοποιηθεί στην
//αξιολόγηση.

        dmg = damage

//Διαχείριση των κινήσεων αλλαγής στατιστικών του βοηθού. Ενεργοποίηση του μετρητή
που αντιστοιχεί στην κίνηση.
Αλλιώς αν (i>=4 & i<7):

        Counters[i-3] = 1

//Διαχείριση των κινήσεων επαναφοράς πόντων ζωής του βοηθού.

Αλλιώς αν (i = 7):

        Player.CHP = Player.CHP + 4

//Αποθηκεύουμε αν εκτελέστηκε κίνηση επαναφοράς πόντων ζωής ώστε να χρησιμοποιηθεί
//στην αξιολόγηση.

        heal = 1

Αλλιώς:

        Helper.CHP = Helper.CHP + 4

        heal = 2


Επέστρεψε CHP_Characters, Counters[], heal, dmg
```

Αλγόριθμός 4.2: Συνάρτηση helperAction()

```
//Παρόμοια λογική με τη συνάρτηση helperAction()
Δήλωση attackPattern, moveNumber, counterState, possibility, damageId
Αν (j = 0):
        Counters[4] = 1
Αλλιώς αν (j = 3 & chosenEnemy = 4):
        EnemyCHP = EnemyCHP + 8
Αλλιώς:
        attackPattern = chosenEnemy.ToString() + H
```

```
        Αν (j = 2 | j = 4 | j = 6)
                attackPattern = chosenEnemy.ToString() + P
        moveNumber = enemyMoves[j]
        counterState = Counters[0].toString() + Counters[1].ToString() +
Counters[2].ToString() + Counters[3].ToString() + Counters[4].ToString()
        possibility = 1
        Αν (j =5 | j = 6):
                possibility = 2
        damageId = attackPattern + moveNumber + counterStates + possibility
        dmg = SearchDmgCatalogue(damageId)

        Αν (j= 1 | j = 3 | j =5):
                Helper.CHP = Helper.CHP – dmg
        Αλλιώς:
                Player.CHP = Player.CHP – dmg

        Επέστρεψε CHP_Characters, Counters[]
```

Αλγόριθμος 4.3: Συνάρτηση enemyAction()

```
//Ελέγχει αν η κίνηση μπορεί να εκτελεστεί, εξετάζοντας τον αντίστοιχο μετρητή.
Αν (i =0):
        Αν (Counters[4] = 1):
                Επέστρεψε 'ΑΛΗΘΗΣ'
Αλλιώς αν (i>0):
        Αν (Counters[i-3] = 1):
                Επέστρεψε 'ΑΛΗΘΗΣ'
```

Αλγόριθμος 4.4: Συνάρτηση checkCounter()

4.3. Συνάρτηση Αξιολόγησης

Το τελευταίο στοιχείο του αλγορίθμου που πρέπει να οριστεί είναι η συνάρτηση αξιολόγησης. Σκοπός της συνάρτησης αυτής είναι να αποδίδει μια αριθμητική αξία στις τελικές καταστάσεις (φύλλα) του δένδρου αναζήτησης και να προσδιορίζει το πόσο ευνοϊκές είναι για τον παίκτη με βάση συγκεκριμένα κριτήρια.

Ο ρόλος του βοηθού στο παιχνίδι είναι η υποστήριξη του παίκτη με στόχο την επιβίωση του. Συνεπώς, η στρατηγική του βασίζεται στα ακόλουθα κριτήρια, τα οποία παρουσιάζονται με σειρά προτεραιότητας:

1. Η άμεση νίκη του παίκτη.

2. Η επιβίωση του παίκτη.
3. Η επιβίωση του ίδιου.
4. Η αποτελεσματικότητα των ενεργειών του (ζημιά ή επαναφορά πόντων ζωής) με κριτήριο την τρέχουσα κατάσταση της μάχης.
5. Η ενδυνάμωση του παίκτη και η αποδυνάμωση του αντιπάλου.

Με βάση τις προτεραιότητες αυτές, ορίζεται ένα σύστημα βαθμολόγησης το οποίο αποδίδει αριθμητικές τιμές στις διάφορες παραμέτρους μιας κατάστασης του παιχνιδιού:

- **Βασική Βαθμολογία (Πόντοι Ζωής):**
    - Οι τρέχοντες πόντοι ζωής του παίκτη κυμαίνονται μεταξύ 21-30: **+ 100 πόντοι.**
    - Οι τρέχοντες πόντοι ζωής του παίκτη κυμαίνονται μεταξύ 11-20: **+ 50 πόντοι.**
    - Οι τρέχοντες πόντοι ζωής του παίκτη κυμαίνονται μεταξύ 1-10: **+ 10 πόντοι.**
    - Οι τρέχοντες πόντοι ζωής του παίκτη είναι 0 (**Ήττα**): **-1000 πόντοι.**
    - Οι τρέχοντες πόντοι ζωής του βοηθού κυμαίνονται μεταξύ 21-30: **+ 25 πόντοι.**
    - Οι τρέχοντες πόντοι ζωής του βοηθού κυμαίνονται μεταξύ 11-20: **+ 15 πόντοι.**
    - Οι τρέχοντες πόντοι ζωής του βοηθού κυμαίνονται μεταξύ 1-10: **+5 πόντοι.**
    - Οι τρέχοντες πόντοι ζωής του βοηθού είναι 0: **-50 πόντοι.**
    - Οι τρέχοντες πόντοι ζωής του αντιπάλου κυμαίνονται μεταξύ 66%-100% των συνολικών: **+0 πόντοι.**
    - Οι τρέχοντες πόντοι ζωής του αντιπάλου κυμαίνονται μεταξύ 33%-65% των συνολικών: **+ 20 πόντοι.**
    - Οι τρέχοντες πόντοι ζωής του αντιπάλου κυμαίνονται μεταξύ 1-32% των συνολικών: **+50 πόντοι.**
    - Οι τρέχοντες πόντοι ζωής του αντιπάλου είναι 0 (**Νίκη**): **+1000 πόντοι.**
- **Τακτικά Μπόνους:**
    - **Μπόνους Ζημιάς:** Η συνάρτηση επιβραβεύει τις επιθέσεις που προκαλούν μεγάλη ζημιά. Το μπόνους αυτό προσδιορίζεται από έναν πολλαπλασιαστή, ο οποίος εφαρμόζεται στην ζημιά της κίνησης. Ο πολλαπλασιαστής αυτός τροποποιείται ανάλογα με την κατάσταση του παιχνιδιού: μειώνεται όταν ο παίκτης ή ο βοηθός έχουν λίγους πόντους ζωής και αυξάνεται όταν ο αντίπαλος είναι σε κρίσιμη κατάσταση. Με αυτό το τρόπο δίνεται προτεραιότητα στην επιβίωση.
    - **Μπόνους Επαναφοράς Πόντων Ζωής:** Αυτό το μπόνους εφαρμόζεται όταν ο παίκτης και ο βοηθός έχουν χαμηλούς πόντους ζωής. Όσο πιο χαμηλά είναι οι πόντοι τους τόσο πιο μεγάλο είναι το μπόνους, καθιστώντας τις κινήσεις επαναφοράς πόντων ζωής προτεραιότητα σε κρίσιμες καταστάσεις.
    - **Μπόνους Μετρητών:** Πέρα από ένα βασικό μπόνους για κάθε ενεργό μετρητή που υπάρχει στο παιχνίδι (που σηματοδοτεί ότι μια κίνηση αλλαγής στατιστικών ενεργεί στο παιχνίδι), η συνάρτηση αναγνωρίζει και επιβραβεύει συνδυασμούς. Η ταυτόχρονη ενεργή κατάσταση συγκεκριμένων ζευγών μετρητών δίνει επιπλέον πόντους, προωθώντας πιο σύνθετες στρατηγικές.

Η τελική βαθμολογία κάθε κατάστασης προκύπτει από το άθροισμα των αντίστοιχων πόντων της.

- Για παράδειγμα, σε μια κατάσταση όπου ο παίκτης έχει 13 CHP, ο βοηθός 6 CHP, ο αντίπαλος 44% CHP και όλοι οι μετρητές είναι ανενεργοί (00000), η συνολική βαθμολογία είναι: 50+5+20+0=75.

- Αντίστοιχα, σε μια κατάσταση όπου ο παίκτης έχει 22 CHP, ο βοηθός 15 CHP, ο αντίπαλος 21% CHP και οι μετρητές έχουν τιμή 11100, η τελική βαθμολογία είναι: 100+15+50+25=190. Η δεύτερη κατάσταση αξιολογείται ως σημαντικά πιο ευνοϊκή από την πρώτη.

Παρακάτω παρουσιάζεται ο αλγόριθμος που εξετάζει τις μεταβλητές της κατάστασης και, βάσει των παραπάνω κανόνων, της αποδίδει την τελική της βαθμολογία:

```
Δήλωση Grade, Sum, dmgMul


Grade = 0
Αν (Player.CHP>20):
        Grade = Grade + 100
Αλλιώς αν (Player.CHP > 10 & Player.CHP < 21):
        Grade = Grade + 50
Αλλιώς αν (Player.CHP>0 & Player.CHP < 11):
        Grade = Grade + 10
Αλλιώς:
        Grade = Grade – 1000


Αν (Helper.CHP>20):
        Grade = Grade + 25
Αλλιώς αν (Helper.CHP > 10 & Helper.CHP < 21):
        Grade = Grade + 15
Αλλιώς αν (Helper.CHP > 0 & Helper.CHP < 11):
        Grade = Grade + 5
Αλλιώς:
        Grade = Grade – 50


Αν (Enemy.CHP > (Enemy.HP*0.33) & Enemy.CHP < (Enemy.HP*0.66)):
        Grade = Grade + 20
Αλλιώς αν (Enemy.CHP > 0 & Enemy.CHP < (Enemy.HP*0.33)):
        Grade = Grade + 50
Αλλιώς αν (Enemy.CHP = 0):
        Grade = Grade + 1000


Sum = 0
```

```
//Ο τελευταίος μετρητής δεν ελέγχεται αφού δεν ενισχύει τον παίκτη ούτε
αποδυναμώνει τον αντίπαλο

//Καταμέτρηση των ενεργών μετρητών. Οι μετρητές 0-3 επηρεάζουν την αξιολόγηση της
//κατάστασης, οπότε ο 4ος (του αντιπάλου) δεν αξιολογείται.

Για (i=0) επανέλαβε όσο (i<4):

        Αν (Counters[i] > 0):

                Sum = Sum + 1

//Μπόνους για συγκεκριμένους συνδυασμούς ενεργών μετρητών

Αν (Counters[0] == 1 & Counters[1] = 1):

        Grade = Grade + 10

Αν (Counters[2] == 1 & Counters[3] = 1):

        Grade = Grade + 10

Αν ((Counters[0]=0 & sum = 1)) | (Counters[0]=1 & sum =2)):

        Grade = Grade + 17

//Γενικό μπόνους ανάλογα με το πλήθος των ενεργών μετρητών

Αν (sum>0 & sum<4):

        Grade = Grade + sum*5

Αλλιώς:

        Grade = Grade + 15


//Τακτικό Μόνους Ζημιάς (dmg)

Αν (state2.dmg > 0):

        dmgMul = 0.4

        Αν (Player.CHP>0 & Player.CHP < 11):

                dmgMul = dmgMul*0.01

        Αν (Helper.CHP>0 && Helper.CHP <11):

                dmgMul = dmgMul*0.5

        Αν (Enemy.CHP < (Enemy.HP*0.33)):

                dmgMul = dmgMul*3

        Αλλιώς Αν (enemy.CHP <= (Enemy.HP*0.66)):

                dmgMul = dmgMul*1.5

        Grade = Grade + ΣτρογγυλοποιημένοΠροςΤαΚάτω(State2.dmg*dmgMul)

//Τακτικό μπόνους επαναφοράς πόντων ζωής

Αν (state2.heal = 1):

        Αν (Player.CHP >0 & Player.CHP < 11):

                Grade = Grade + 20

        Αλλιώς Αν (Player.CHP >10 & Player.CHP < 21):

                Grade = Grade + 5

Αλλιώς Αν (state2.heal =2):

        Αν (Helper.CHP >0 & Helper.CHP <11):

                Grade = Grade + 10


Επέστρεψε Grade
```

Αλγόριθμος 4.5: Συνάρτηση Αξιολόγησης

4.4. Υλοποίηση του Αλγορίθμου σε Κώδικα

Στην παρούσα ενότητα, οι αλγόριθμοι που αναλύθηκαν προηγουμένως με μορφή ψευδοκώδικα υλοποιούνται σε γλώσσα προγραμματισμού C#, με σκοπό την ενσωμάτωση τους στο παιχνίδι. Θα παρουσιαστεί μια απλοποιημένη εκδοχή του κώδικα που εστιάζει στην καλύτερη κατανόηση του αλγορίθμου.

Αρχικά, για την αποτελεσματικότερη διαχείριση των δεδομένων που ορίζουν μια κατάσταση του παιχνιδιού, δημιουργείται μια κλάση με όνομα *gameState*. Η κλάση αυτή αποτελεί το βασικό δομικό στοιχείο του αλγορίθμου, καθώς κάθε κόμβος του δένδρου αναζήτησης αναπαρίσταται από ένα αντικείμενο αυτού του τύπου.

```
public partial class gameState : Node {
//Δήλωση των ιδιωτικών πεδίων της κλάσης.
        private int _playerHP;
        …
        private int[] counterState;
        …
        private int _heal;
//Δήλωση των δημόσιων ιδιοτήτων της κλάσης.
        public int playerHP{
                get { return _playerHP; }
                set { _playerHP = value; }
        }
        …
//Κατασκευαστής για την δημιουργία και αρχικοποίηση αντικειμένων τύπου gameState
        public gameState(int PlayerHP, int HelperHP, …){
                playerHP = PlayerHP;
                …
//Δημιουργείται ένα ανεξάρτητο αντίγραφο του πίνακα μετρητών ώστε οι αλλαγές σε
//αυτόν στη νέα κατάσταση να μην επηρεάζουν την κατάσταση από την οποία προήλθε.
                counterState = (int[])CounterState.Clone();
                …
                heal = Heal;
        }
```

```
//Δημιουργεί ένα νέο, ανεξάρτητο αντίγραφο της τρέχουσας κατάστασης, καλώντας τον
//κατασκευαστή.
        public gameState Clone(){
                return new gameState (this.playerHP, this.helperHP, …);
        }
```

Κώδικας 4.1: Κλάση gameState

Για την υλοποίηση της κεντρικής λογικής του αλγορίθμου, δημιουργείται μια κλάση με το όνομα *movePrediction*. Κατά την αρχικοποίηση της, η κλάση αυτή δέχεται ως παραμέτρους την τρέχουσα κατάσταση των τριών χαρακτήρων της μάχης, τους μετρητές, καθώς και τους πίνακες που περιέχουν τα αναγνωριστικά των διαθέσιμων κινήσεων του βοηθού και των αντιπάλων. Η κύρια μέθοδος της κλάσης, *chooseBestAction*(), εκτελεί την ανάλυση του δένδρου παιχνιδιού και αφού ολοκληρώσει τους υπολογισμούς της, επιστρέφει το αναγνωριστικό της βέλτιστης κίνησης. Αυτός ο ακέραιος ανατίθεται στη συνέχεια τη μεταβλητή *chosenAction* που καθορίζει την ενέργεια του βοηθού στην τρέχουσα σειρά του.

```
public partial class movePrediction : Node {
        private character _player;
        …
        private int [] _enemyActions;

        public character player {
                get { return _player; }
                set { _player = value; }
        }
        …
//Αντικείμενο για την παραγωγή τυχαίων αριθμών.
        private Random rng = new Random();

//Βοηθητική δομή (struct) για την αποθήκευση της βαθμολογίας κάθε κίνησης.
        public struct moveScore {
                public int index { get; set; }
                public int finalGrade { get; set;}
        }

//Κατασκευαστής που αρχικοποιεί την κλάση με τα δεδομένα της μάχης.
        public movePrediction(character Player, …){
                Player = Player;
                …
        }
```

```
//Η κύρια μέθοδος του αλγορίθμου που επιστρέφει το αναγνωριστικό της βέλτιστης
//κίνησης.
	public int chooseBestAction(){
		int i,k,…,grade;
//Λίστα για τις βαθμολογίες
		var moveScores = new List<moveScore>();
		bool flag;
/Δημιουργία της αρχικής κατάστασης (ρίζα).
		gameState state0 = new gameState(player.CHP, helper.CHP, enemy.CHP,
	counter, 0, 0);


//Βρόχος που αξιολογεί κάθε πιθανή κίνηση(i) του βοηθού.
		for(i=0; i<9; i++){
//Παράλειψη κίνησης στατιστικών αν ο αντίστοιχος μετρητής είναι ήδη ενεργός.
			if(i>3 && i<7){
				flag = checkCounter(i, state0.counterState);
				if (flag == true){
					continue;
				}
			}
//Δημιουργία κατάστασης μετά την κίνηση του βοηθού (κόμβος βάθους 1).
			gameState state1 = state0.Clone();
			helperAction(state1, chosenEnemy, helperActions, i);

			number=0;
			sum=0;
			if(chosenEnemy>0 && chosenEnemy <4){
				k=7;
			}else{…
//Εσωτερικός βρόχος για αξιολόγηση κάθε πιθανής αντίδρασης(j) του αντιπάλου.
			for(j=0; j<k; j++){
				if(j==0){
				//Παρόμοιος έλεγχος μετρητή για την κίνηση του
			αντιπάλου.
				}
//Δημιουργία κατάστασης (κόμβος-φύλλο) μετά την κίνηση του αντιπάλου.
				gameState state2 = state1.Clone();
				enemyAction(state2, chosenEnemy, enemyActions, j);
//Κλήση της συνάρτησης αξιολόγησης και άθροιση των αποτελεσμάτων.
				grade = Evaluation(state2, enemy);
				number = number +1;
				sum = sum + grade;
			}
```

```
//Υπολογισμός της αναμενόμενης τιμής για την κίνηση 'i'.
                    if (number>0){
                            expectedValue = sum/number;
                    else{
                            expectedValue =0;
                    }
                moveScores.Add(new moveScore{index=i,
                finalGrade=expectedValue});
            }
//Ταξινόμηση των κινήσεων με φθίνουσα σειρά βαθμολογίας.
            var sortedMoves = moveScores.OrderByDescending(move =>
            move.finalGrade).ToList();

            if(sortedMoves.Count >0){
                    maxIndex =0;
//Αν είτε οι 2 είτε οι 3 πρώτες κινήσεις έχουν ίδια βαθμολογία, επιλέγεται μια
//τυχαία.
                    if(sortedMoves.Count>2 && …){
                            maxIndex = rng.Next(0,3);
                    else if (…){…
//Επιστροφή του αναγνωριστικού της κίνησης που επιλέχθηκε.
                    return helperActions[sortedMoves[maxIndex].index];
            }else{
                    return helperActions[0];
            }
        }
}
```

Κώδικας 4.2: Κλάση movePrediction

Η μέθοδος *helperAction()* προσομοιώνει τις εννέα πιθανές κινήσεις του βοηθού. Για κάθε κίνηση, τροποποιεί ένα ανεξάρτητο αντίγραφο της αρχικής κατάστασης, διαμορφώνοντας έτσι τους κόμβους-παιδιά που αναπαριστούν την κατάσταση του παιχνιδιού μετά από κάθε πιθανή ενέργεια. Αντίστοιχα, η μέθοδος *enemyAction()* λαμβάνει αυτούς τους νέους κόμβους και εφαρμόζει τις πιθανές αντιδράσεις του αντιπάλου, δημιουργώντας έτσι τα τελικά φύλλα του δένδρου. Καθώς οι δύο μέθοδοι έχουν σχεδόν πανομοιότυπη λογική, παρουσιάζεται ο κώδικας μόνο της μιας.

```
public void helperAction (gameState state1, int chosenEnemy, int [] helperActions,
int i){
        string attackPattern, moveNumber, counterSt ="", possibility, damageId;
        int j, damage =0;
```

```
//Εύρεση της ζημιάς δημιουργώντας την συμβολοσειρά του αναγνωριστικού της στο
πίνακα damageCatalogue και αφαίρεση της από το τους πόντους ζωής του αντιπάλου για
να προκύψει η νέα κατάσταση.
        if (i>=0 && i<4){
                attackPattern = "H" + chosenEnemy.ToString();
                moveNumber = helperActions[i].ToString();

                for (j=0; j<5; j++){
                        counterSt = counterSt + state1.counterState[j].ToString();
                }

                possibility = "1";
                damageId = attackPattern + moveNumber + counterSt + possibility;

                damage = searchDmgCatalogue(damageId, "Το Path του JSON αρχείου
                damageCatalogue");
                state1.enemyHP = state1.enemyHP - damage;
//Αποθήκευση της τιμής της ζημιάς.
                state1.dmg = damage;
//Ενεργοποίηση του μετρητή της αντίστοιχης κίνησης στατιστικών για να προκύψει η
νέα κατάσταση.
        }else if(i>=4 && i<7){
                state1.counterState[i-3] = 1;
//Επαναφορά πόντων ζωής στον αντίστοιχο χαρακτήρα.
        }else if (i == 7){
                state1.playerHP = state1.playerHP + 4;
//Αποθήκευση της ταυτότητας του χαρακτήρα που δέχτηκε επαναφορά στους πόντους ζωής
του.
                state1.heal = 1;
        }else if (i==8){
                state1.helperHP = state1.helperHP + 4;
                state1.heal = 2;
        }
}
```

Κώδικας 4.3: Μέθοδος helperAction()

Η μέθοδος *Evaluation()* είναι η συνάρτηση αξιολόγησης που βαθμολογεί τα φύλλα του δένδρου ώστε να υπολογίσει την βαθμολογία τους.

```
public int Evaluation(gameState state2, character enemy){
	int grade=0, sum=0, i;
//Αξιολόγηση της κατάστασης βάσει των πόντων ζωής κάθε χαρακτήρα.
	if (state2.playerHP > 20){
		grade = grade + 100;
	else if…

	if (state2.helperHP > 20){
		grade = grade + 25;
	else if…

	if (state2.enemyHP > (enemy.HP*0.33) && state2.enemyHP <= (enemy.HP*0.66) {
		grade = grade + 20;
	else if…
//Καταμέτρηση των ενεργών μετρητών και επιβράβευση με βάση το πλήθος τους.
	for (i=0; i<4; i++){
		if (state2.countersState[i] >0) {sum = sum + 1;}
	}
	if (sum>0 && sun<4){
		grade = grade + sum*5;
	else {
		grade = grade + 15;
	}
//Ειδική βαθμολόγηση με βάση τον συνδυασμό ενεργών μετρητών.
	if (state2.countersState[0] == 1 && state2.counterState[1] == 1){
		grade = grade + 10;
	}
	…
//Υπολογισμός ενός τροποποιητή ζημιάς με βάση την κατάσταση του παιχνιδιού. Η ζημιά
//πολλαπλασιάζεται με αυτόν για να προστεθεί ως μπόνους στην τελική βαθμολογία.
	if (state2.dmg >0){
		float dmgMul = 0.4f;

		if (state2.playerHP > 0 && state2.playerHP < 11) {dmgMul*0.01f;}
		…
		grade = grade + (int)Math.Floor(state2.dmg*dmgMul);
	}
```

```
//Αξιολόγηση του πόσο ωφέλιμη είναι μια κίνηση επαναφοράς πόντων ζωής στην
//κατάσταση του παιχνιδιού και κατάλληλη βαθμολόγηση.

        if (state2.heal == 1){

                if (state2.playerHP >0 && state2.playerHP<11) {grade = grade+20;}

                else if…

        else if (state2.heal == 2) {…


        return grade;

}
```

Κώδικας 4.4: Μέθοδος evaluation()

Η μέθοδος *seachDmgCatalogue()* αναζητεί τον κατάλογο ζημιών με το αναγνωριστικό ζημιάς που δημιουργήθηκε από τις μεθόδους *helperAction()* και *enemyAction()* και επιστρέφει την αντίστοιχη αριθμητική τιμή της ζημιάς.

```
public int searchDmgCatalogue(string damageId, string fileName){

        int dmg;


        string jsonString;

//Πραγματοποιείται πρόσβαση στο αρχείο που ορίζεται από τη μεταβλητή 'fileName' σε
//κατάσταση ανάγνωσης. Η χρήση της εντολής using διασφαλίζει το κλείσιμο του μετά
//την ανάγνωση. Το περιεχόμενο του αποθηκεύεται ως μια συμβολοσειρά στη μεταβλητή
//'jsonString'.

        using (var = FileAccess.Open(fileName, FileAccess.ModeFlags.Read)){

                JsonString = file.GetAsText();

        }

//Η συμβολοσειρά jsonString αποθηκεύεται ως μια δομή δεδομένων τύπου JsonNode με
όνομα catalogueNode.

        JsonNode catalogueNode = JsonNode.Parse(jsonString);

//Γίνεται έλεγχος αν το περιεχόμενο της μεταβλητής catalogueNode είναι αντικείμενο
JSON.

        if(catalogueNode is JsonObject catalogueObject){

//Γίνεται έλεγχος αν το συγκεκριμένο αναγνωριστικό ζημιάς υπάρχει ως κλειδί στον
//κατάλογο.

                if(catalogueObject.ContainsKey(damageId)){

//Ανάκτηση της αριθμητικής τιμής της ζημιάς που αντιστοιχεί στο συγκεκριμένο
κλειδί.

                        dmg = (int)catalogueObject[damageId];

                        return dmg;
```

```
            else {
//Αν ένα βασικό αναγνωριστικό ζημιάς (baseDamageId) δεν βρεθεί, πρόκειται για
σφάλμα στα δεδομένα, οπότε επιστρέφετε 0.
                  if (damageId.EndsWith("000001") ||
            damageId.EndsWith("000002")){
                        return 0;
                  }
//Αν το αναγνωριστικό δεν υπάρχει στο κατάλογο, αφαιρούνται οι πληροφορίες των
μετρητών για να αναζητηθεί η βασική ζημιά(baseDamageId) μέσω αναδρομικής κλήσης.
                  int startIndex = damageId.Length - 6;
                  string temp = damageId.Remove(startIndex, 5);
                  string baseDamageId = temp.Insert(startIndex, "00000");
                  dmg = searchDmgCatalogue(baseDamageId, fileName);
                  return dmg;
      }
}
```

Κώδικας 4.5: Μέθοδος searchDmgCatalogue()

Τέλος, η μέθοδος *checkCounter()* ελέγχει ποιοι μετρητές είναι ενεργοί για να καθορίσει ποιες καταστάσεις μπορούν να δημιουργηθούν και να αξιολογηθούν.

```
public book checkCounter(int i, int [] counterState){
      if(i == 0){
            if(counterState[4] == 1){
                  return true;
            }
      }else{
            if (counterState[i-3] == 1){
                  return true;
            }
      }
      return false;
}
```

Κώδικας 4.6: Μέθοδος checkCounter()

Κεφάλαιο 5- Συμπεράσματα

5.1. Συμπεράσματα

Η παρούσα πτυχιακή εργασία είχε δύο στόχους: την ανάπτυξη ενός ηλεκτρονικού παιχνιδιού και την ενσωμάτωση ενός συστήματος επιλογής κινήσεων.

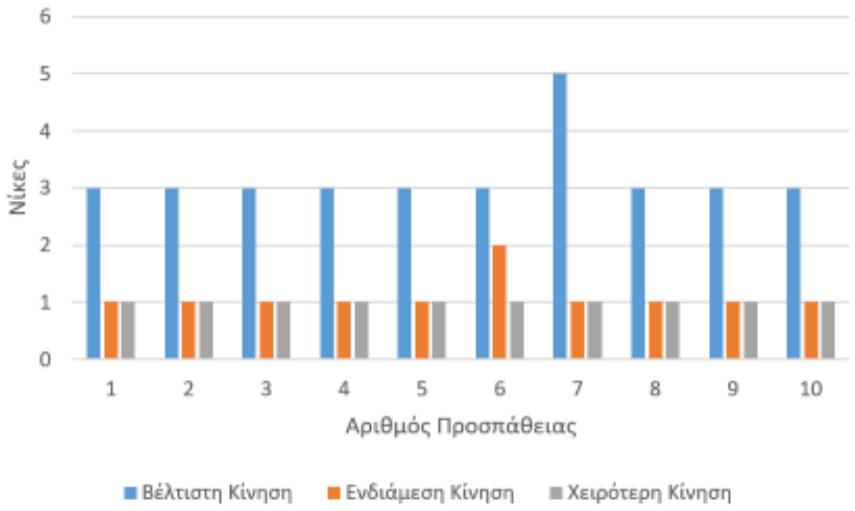
Αναφορικά με τον πρώτο στόχο, παρόλο που οι περιορισμοί του έργου δεν επέτρεψαν την ανάπτυξη ενός πλήρους ηλεκτρονικού παιχνιδιού ρόλων, το τελικό αποτέλεσμα ανταποκρίνεται στα κριτήρια που ορίζουν ένα ηλεκτρονικό παιχνίδι (*βλ. ενότητα 1.2*). Συγκεκριμένα, το πρωτότυπο που σχεδιάστηκε και υλοποιήθηκε είναι πλήρως λειτουργικό και πληροί όλες τις απαραίτητες προϋποθέσεις, καθιστώντας τον πρώτο στόχο επιτυχώς ολοκληρωμένο.

Για την αξιολόγηση του συστήματος επιλογής κινήσεων, σχεδιάστηκε ένα πείραμα μέτρησης της απόδοσης του. Το πειραμά ορίστηκε ως εξής:

- Ο παίκτης και ο βοηθός αντιμετωπίζουν διαδοχικά όλους τους αντίπαλους του παιχνιδιού.
- Κάθε πλήρης κύκλος μαχών μέχρι την ήττα του παίκτη ορίζεται ως μια "προσπάθεια".
- Μετά από κάθε νίκη, η ομάδα προχωρά στον επόμενο αντίπαλο διατηρώντας τους πόντους ζωής που απέμειναν από την προηγούμενη μάχη.
- Το μέτρο απόδοσης είναι το πλήθος των αντιπάλων που ηττήθηκαν εντός μιας προσπάθειας.
- Σε περίπτωση ήττας του βοηθού, αυτός επανέρχεται στην επόμενη μάχη με 1 πόντο ζωής (*CHP*).
- Ο παίκτης εκτελεί πάντα την ίδια προκαθορισμένη κίνηση (Sword Slash).

Το πείραμα εκτελέστηκε τρεις φορές, μεταβάλλοντας μόνο τη στρατηγική του βοηθού: μια εκτελώντας τη βέλτιστη κίνηση (όπως υπολογίστηκε από το σύστημα), μια την κίνηση ενδιάμεσης απόδοσης και μια τη χειρότερη. Για κάθε μια από τις τρεις στρατηγικές, πραγματοποιήθηκαν 10 προσπάθειες.

Τα αποτελέσματα του πειράματος παρουσιάζονται στην *εικόνα 5.1*.



Εικόνα 5.1: Αποτελέσματα πειράματος.

Από την ανάλυση των αποτελεσμάτων, προκύπτει το συμπέρασμα ότι η επιλογή μεταξύ της χειρότερης και της ενδιάμεσης κίνησης παρουσιάζει αμελητέα διαφορά στην απόδοση. Συγκεκριμένα, και οι δύο στρατηγικές οδήγησαν, στη πλειοψηφία των προσπαθειών, σε μια μόνο νίκη. Εξαίρεση αποτελεί η έκτη προσπάθεια της ενδιάμεσης στρατηγικής, στην οποία η ομάδα κατάφερε να επιτύχει δύο νίκες πριν την ήττα της.

Αντιθέτως, τα αποτελέσματα από τη χρήση της βέλτιστης κίνησης επιβεβαιώνουν την αποτελεσματική λειτουργία του συστήματος επιλογής. Η στρατηγική αυτή εξασφάλισε τουλάχιστον τρεις νίκες σε κάθε προσπάθεια, γεγονός που αποδεικνύει τη σημαντικά βελτιωμένη απόδοση της σε σχέση με τις άλλες δύο στρατηγικές. Ειδικότερα, η έβδομη προσπάθεια δείχνει την ικανότητα του συστήματος να ενισχύει σημαντικά την απόδοση, επιτρέποντας την επίτευξη πολλαπλών νικών ακόμη και όταν ο παίκτης ακολουθεί μια σταθερή, μη στρατηγική προσέγγιση.

5.2. Μελλοντικές Επεκτάσεις

Το πρωτότυπο λογισμικού που υλοποιήθηκε στην παρούσα εργασία αποτελεί την πρώτη εκδοχή του έργου. Σύμφωνα με την επαναληπτική μεθοδολογία ανάπτυξης ("Κτίσε, Αξιολόγησε, Επανέλαβε"), μετά την ολοκλήρωση και την αξιολόγηση ενός πρωτοτύπου, ακολουθεί η ανάπτυξη του επόμενου, το οποίο ενσωματώνει επιπρόσθετα στοιχεία του τελικού έργου. Συνεπώς, παρατίθενται ορισμένες προτάσεις που μπορούν να αποτελέσουν τη βάση για μελλοντικές επεκτάσεις του πρωτοτύπου.

Στο επίπεδο του παιχνιδιού, προτείνεται η ενσωμάτωση:

- Ενός κόσμου διαθέσιμου προς εξερεύνηση.
- Ενός συστήματος προόδου που θα επιβραβεύει τον παίκτη με βελτιωμένα στατιστικά.
- Μεγαλύτερής ποικιλίας τύπων αντιπάλων.
- Ενός συστήματος παραμετροποίησης χαρακτήρων, που θα επιτρέπει στον παίκτη να ορίσει τα στατιστικά και τις κινήσεις του.
- Μιας κεντρικής αφήγησης που θα πλαισιώνει το σύστημα μάχης.

Αναφορικά με το σύστημα επιλογής κινήσεων, προτείνονται οι ακόλουθες επεκτάσεις:

- Ο συνυπολογισμός των πιθανών κινήσεων του παίκτη κατά τον υπολογισμό της βέλτιστης κίνησης του βοηθού.
- Η αύξηση του βάθους του δένδρου παιχνιδιού, επιτρέποντας στον αλγόριθμο να προβλέπει πιο στρατηγικά τις κινήσεις του.
- Η ανάπτυξη ενός παράλληλου, συμβουλευτικού συστήματος για τον παίκτη, το οποίο θα προτείνει κινήσεις αντί να τις εκτελεί αυτόματα.

ΒΙΒΛΙΟΓΡΑΦΙΑ

[1] Salen, K., & Zimmerman, E. (2004). *Rules of Play: Game Design Fundamentals*. The MIT Press.

[2] Donavan, T. (2010). *Replay: The history of video games*. Yellow Ant.

[3] Rollings, A. & Adams, E. (2003). *Andrew Rollings and Ernest Adams on game design*. New Riders.

[4] Barton, M. (2008). *Dungeons and desktops: The history of computer role-playing games*. A K Peters/CRC Press

[5] Pepe, F. (Ed.). (2019). *The CRPG book: A guide to computer role-playing games (Expanded ed.)*. Bitmap Books

[6] Adams, E. (2014). Fundamentals of game design (3rd ed.). New Riders.

[7] Yang, Y. (2023). *Foundations of Game Design (IAT 312 D100) [Course outline]*. School of Interactive Arts and Technology, Simon Fraser University https://www.sfu.ca/outlines.html?2023/fall/iat/312/d100

[8] Weeks, M. (2024). *4821/6821- Fundamentals of Game Programming [Syllabus]*. Department of Computer Science, Georgia State University https://hallertau.cs.gsu.edu/~mweeks/csc4821/

[9] Kramarzewski, A. , & De Nucci, E. (2018). *Practical Game Design*. CRC Press.

[10] Adams, E., & Dormans, J. (2012). *Game Mechanics: Advanced Game Design*. New Riders

[11] The Godot Project (n.d.). *Godot Engine documentation*. https://docs.godotengine.org/en/stable/index.html

[12] Levitin, A. (2012). *Introduction to the design and analysis of algorithms (3rd ed.)*. Pearson.

[13] Russel, S. J., & Norvig, P. (2020). *Artificial Intelligence: A modern approach (4th ed.)*. Pearson.

[14] Banks, J., Carson, J. S. II, Nelson, B. L., & Nicol, D. M. (2010). *Discrete-event system simulation (5th ed.)*. Pearson.

[15] Walpole, R.E., Myers, S. L., & Ye, K. (2012). *Probability & statistics for engineers & scientists (9th ed.)*. Pearson.

[16] Aho, A. V., Hopcroft, J. E., & Ullman, J. D. (1983). *Data Structures and Algorithms*. Addison-Wesley.

[17] Pearl, J. (1984). *Heuristics: Intelligent search strategies for computer solving*. Addison-Wesley.

[18] Marshall, C. (2018, October 19). *The making of Undertale. PC Gamer*. https://www.pcgamer.com/the-making-of-undertale/

[19] Bray, T. (Ed.). (2017). *The JavaScript Object Notation (JSON) data interchange format (RFC 8259)*. Internet Engineering Task Force. https://datatracker.ietf.org/doc/html/rfc8259

[20] Wizards of the Coast. (2014). *Player's Handbook (5th ed.)*. Wizards of the Coast.